NETCARBENCH: A benchmark for techniques and tools used in the design of automotive communication systems

Christelle Braun, Lionel Havet, Nicolas Navet

NETCARBENCH: A BENCHMARK FOR TECHNIQUES AND TOOLS USED IN THE DESIGN OF AUTOMOTIVE COMMUNICATION SYSTEMS

Christelle Braun, Lionel Havet, Nicolas Navet

LORIA-INRIA
Campus Scientifique, B.P. 239
54506 Vandoeuvre-lès-Nancy, France
{Christelle.Braun,Lionel.Havet,Nicolas.Navet}@loria.fr

Abstract: This paper presents NETCARBENCH, a benchmark devoted to improve the assessment, the understanding and the comparability of techniques and tools used in the design of in-vehicle communication networks. This benchmark is motivated by the increasing use of algorithms intended to optimize the resource utilization in the design and configuration of automotive communication systems. For instance, typical objectives are the minimization of the network bandwidth usage and the reduction of the worst case response times of the frames. The main contribution of NETCARBENCH is to allow a fine-grained user-defined parameterization of the generated message sets by means of XML-configuration files that specify the characteristics of the message sets and the variability thereof. Experiments suggest that the outcomes of NETCARBENCH are satisfactory in terms of their closeness to the input specifications. NETCARBENCH and its user manual are freely available under the GNU General Public License.

1. INTRODUCTION

Context. With the extensive use of electronics, in-vehicle embedded systems have strongly raised in complexity, involving an increasing number of nodes whose communication is typically subject to real-time constraints and high performance requirements. In today's luxury cars, up to 2500 signals (i.e., elementary information such as the speed of the vehicle) are exchanged between ECUs (Electronic Control Units) organized in networks whose size can exceed 70 nodes (Navet *et al.*, 2005).

The strong cost pressure in the automotive industry requires the hardware resources to be used at the fullest of their capacity. On the other hand, several embedded functions are critical from the safety point of view and thus necessitate a guaranteed quality of service (QoS) from the underlying hardware and software platform. An additional requirement of the communication system is to ensure a temporal interoperability between ECUs, since most of them are developed by third-party suppliers.

As a consequence, car manufacturers are starting to design the communication system with the help of optimization tools that are either developed in-house or by third-party suppliers (INRIA, 2006; SymtaVision, 2007). The efficiency of the underlying configuration algorithms will directly impact the safety of the vehicle. In addition, such software tools enable to optimize the resource utilization of both the networks and the microprocessors and, thus, they contribute to reduce the overall hardware costs. In order to evaluate the effectiveness of the tools and to be able to make comparisons, a domain specific benchmark is needed. This is what is addressed by this study, with a focus on Controller Area Network (CAN) (ISO/DIS 11898, 1992), which is a de-facto standard for in-vehicle communications.

Objective. The aim of NETCARBENCH is to generate workloads that closely mimic the data exchange between the ECUs involved in an automotive communication system. The benchmark has therefore to be sufficiently flexible to realistically model the workload on the different types of networks currently embedded in cars [1] .

Furthermore, it should be possible to define the parameters in such a way that the benchmark can adapt to future evolutions and thus be used in the long run. In particular, the benchmark should be scalable regarding modifications in the type or amount of data exchanged between the nodes.

Existing benchmarks. Because of confidentiality and the competition between car manufacturers, very little information has been published concerning benchmarks. In practice, they indeed consist of the real sets of messages exchanged in cars and can therefore not be disclosed. To the best of our knowledge, the only two available benchmarks are the SAE benchmark and the PSA benchmark. They have both been used numerous times for assessing the performance of communication-related techniques.

The *SAE Benchmark* comes from a SAE report (SAE, 1994) describing the set of signals exchanged on point-to-point links in a prototype electric car, and has been later adapted to CAN in (Tindell and Burns, 1994). The benchmark is comprised of 7 subsystems exchanging 53 messages. However, the characteristics of the SAE benchmark are not exactly the ones of a typical automotive CAN network and it is no more realistic in terms of data exchanged and number of nodes. Nowadays, the sole *body network* of a vehicle can be made of about 15 ECUs exchanging 60-80 frames.

The PSA (PSA Peugeot-Citroën) benchmark (Navet *et al.*, 2000) is the set of messages of a *powertrain network* implemented in a prototype car. The network is made of 6 nodes, while the message set consists of 12 different periodic messages. In contrast to SAE benchmark, PSA benchmark was initially designed for CAN, but its characteristics have now become obsolete with respect to a current typical implementation.

Contribution. We specify a new benchmark for broadcast networks in automotive distributed control systems, called NETCARBENCH. It generates sets of messages, similar to the aforementioned existing benchmarks, however it is designed as a parametric benchmark that accepts as input a user-defined configuration. The generation process is performed automatically, so that the user is relieved from any programming burden. Because automotive communication systems often have relatively similar message sets, we provide along with NETCARBENCH several predefined realistic message sets that can be directly used for testing.

Organization. Section 2 presents the technical problems that need to be addressed when configuring the communication. Section 3 gives details about the different steps performed by NETCARBENCH to translate the user-defined configuration into message sets complying with the specified properties. The outcomes of the benchmark are evaluated in Section 4 for the configuration of a typical *body network*. Finally, Section 5 provides some concluding remarks.

2. CONFIGURATION ALGORITHMS TARGETED BY THE BENCHMARK

The goal of our benchmark is to evaluate the performance of algorithms employed for the configuration of the communication between nodes in automotive embedded networks. In these networks, the set of ECUs is known, as well as the set of signals [2] that are to be sent over the network for each ECU (i.e. the size, deadline and production period of the signals). Configuring the communication system involves to build the set of frames transmitted by the ECUs, to assign a unique priority to every frame if a priority bus is used, and to check the feasibility of the resulting system, i.e. check that each signal respects its deadline. This comes to solve two related problems: *frame packing* and *schedulability analysis*.

2.1 Frame packing

Each signal s_i can be characterized by a tuple $(N_i, T_i, O_i, C_i, D_i)$ where:

- N_i is the identifier of the ECU which produces the signal,
- T_i is the *period of production* - generally corresponding to the period of the producing task,
- O_i is the offset of the signal, that is the latest delay after which the first instance of the signal is produced (and subsequent values of the same signal will be released at times $O_i + k \cdot T_i$ at the latest),
- C_i its *size* in bits,
- D_i its *relative deadline* , that is the maximum duration between the production of the signal on the sender side and its reception by all consumers.

[1] A typical in-car embedded system is divided into several functional domains that correspond to different features and constraints. For each functional domain, a distinct network is generally used (e.g. *body*, *chassis*, *powertrain*, *multimedia*,...). The reader may refer to (Navet *et al.*, 2005) for an overview of in-vehicle networking.

[2] In automotive terminology, a signal is an elementary information such as the speed of the vehicle.

The *frame packing* problem consists in constructing a set of frames $F = \{f_1, \ldots, f_k\}$ starting from a given set of signals $S = \{s_1, \ldots, s_n\}$ such that the resulting set of frames on each station is schedulable, i.e. none of the signals transmitted misses its deadline, while utilizing as little bandwidth as possible. Minimizing the bandwidth consumption is important because it enables the use of cheaper components and it facilitates the incremental design process in use in the automotive industry. A *frame* f_i is characterized by the tuple $(N'_i, T'_i, O'_i, C'_i, D'_i, P'_i)$ where N'_i is the identifier of the ECU which sends the frame, T'_i its period, O'_i its offset (i.e., same meaning as previously defined for signals), C'_i its size, D'_i its deadline [3] and P'_i its priority.

In (Norström *et al.*, 2000), the frame packing problem is proven to be NP-hard. In (Saket and Navet, 2006), the exact complexity of the problem is derived and it is shown that an exhaustive approach is not possible even for a very limited number of signals and/or ECUs. A solution is thus to find efficient heuristics such as in (Saket and Navet, 2006), which proved to be more effective than the previously proposed algorithms. However, the literature in this line of work is still limited and, most likely, significant progresses are ahead of us.

2.2 Schedulability analysis

After the frame packing step has been completed, the *schedulability analysis* determines whether the resulting set of frames meets the performance constraints imposed on the system (no deadline miss, jitters on frame reception kept within reasonable bounds, etc). Using the fact that CAN grants the bus access according to the priority of the messages, it is possible to calculate the worst-case response times of the frames (see (Tindell and Burns, 1994), revised in (Davis *et al.*, 2007)) and thus decide upon the feasibility of the system. It is worth pointing out that the problem is more complex when the offsets of the frames are fixed (i.e., when the synchronous case cannot be assumed to be possible), and, to the best of our knowledge, there is no solution whose algorithmic complexity is not exponential.

Finding strategies to reduce the worst-case response time is an important problem. An element of solution is to schedule messages with offsets. This means that the first instance of a frame is transmitted with a delay - the offset - with regard to a reference point, which is the instant at which the station becomes ready to transmit. Subsequent instances of the frame are then sent periodically with the first transmission as time origin. By carefully choosing the offsets, it is possible to desynchronize the traffic streams and avoid the worst-case situation in which a large number of frames are released synchronously. As a result, the workload is distributed more uniformly over time and the response times are reduced, typically by a factor 2 to 4 (see (Grenier *et al.*, 2008) for experimental results).

Two problems, difficult from an algorithmic point of view, are to be solved when implementing offsets: choosing the offsets and computing the worst-case response times. The latter problem is particularly important because the computation might require a large amount of time for systems with a realistic number of messages (e.g., a *body network* with 100 messages having offsets). Since optimization strategies usually involve considering a large number of different configurations, computation time is a crucial performance metric for schedulability assessment algorithms. If the complexity of the system requires to implement an approximation of the worst-case response times, the tightness of the bound is another metric of interest.

This motivates the development of a benchmark that will enable us to better compare the relative performance of the different algorithms and tools, and to further improve them. In our research group, we have a special interest in this objective, since we are developing software (INRIA, 2006) to optimize the resource usage (i.e., CPU and networks) and ease the design and configuration phases.

3. IMPLEMENTATION

In order to generate message sets complying with the specifications of the user, NETCARBENCH first needs to extract the characteristics of the network and the messages from the configuration file. Then, the parameters are instantiated and the generation of the messages is performed. It remains for NETCARBENCH to distribute these messages among the network stations according to the specifications of the user. In the following paragraphs, we address these different steps and describe the solutions implemented in NETCARBENCH in greater detail.

3.1 Format of the generated message set

Given a configuration file (e.g. `body_config.xml` shown in Figure 2) and an integer parameter n, NETCARBENCH generates n message sets which comply with the user-defined specifications, and creates for each set k ($k \leq n$) a new output XML file `set_`k`.xml`.

The characteristics of the frames and signals generated by NETCARBENCH correspond to the model introduced in Section 2.1, with the restrictions that the deadlines are not specified (i.e., typically they are assumed to be equal to the periods $\forall i, D_i = T_i$) and

[3] The deadline of a frame can be deduced from the deadlines of the signals it contains, see (Saket and Navet, 2006) for more details

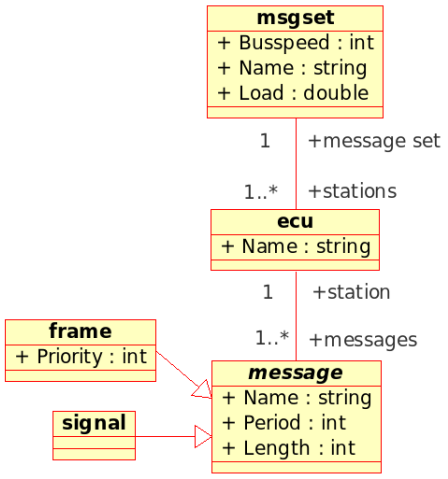

Fig. 1. UML Diagram for the message set in NETCARBENCH

that offsets are not considered. These two features may be implemented in future version of NETCARBENCH.

A UML representation of the format is given in Figure 1 and the listing of a typical file generated by NETCARBENCH is shown in Figure 3. The user has to choose between a message set composed of frames or of signals. This is needed because a frame packing algorithm works on sets of signals, while schedulability analysis handle sets of frames. As shown in Figure 1, the network is modelled as a three-level hierarchy of components: the message set element `msgset` is the root element and the `ecu` elements its children. Each ECU can handle one or more messages, which, for the whole message set, correspond to either `frame` elements or `signal` elements. The components of the model (i.e. ecu, msgset, frame or signal in Figure 1) and their properties (e.g. identifiers, periods, lengths) are then respectively mapped to the XML elements and their attributes in the output file representing the generated message set.

3.2 Network specification

The user should be able to specify, in a precise manner, the characteristics of the networks he wants to generate such that the outputs of NETCARBENCH correspond to what the user wants to model. Also, the parameters which may vary in practice should not be set once for all, so that the benchmark covers as many realistic test-cases as possible. NETCARBENCH supports *parameter randomization*, which means that the parameter values are chosen randomly, with the requirements that these values respect some specified constraints on their definition domain and their probability distribution given as a frequency histogram.

A simple and intuitive XML configuration format has been specified for NETCARBENCH. A typical specification of a *body network* is provided in Figure 2. NETCARBENCH parses this file and sets the parameter values for each of the "artificial" networks generated. A brief summary of the meaning of the parameters is given in the following paragraph - additional information can be found in the user manual of NETCARBENCH (Braun *et al.*, 2007).

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
<load Min="30" Max="35" />
<ecu Min="15" Max ="20" />
<bandwidth Value="125" />
<signals Value="FALSE" />
<periods>
  <p Value="50" Weight="2"
    PrioLowRange="1" PrioHighRange="200"
    Margin="1"/>
    ...
  <p Value="2000" Weight="5"
    PrioLowRange="500" PrioHighRange="1500"
    Margin="1" />
</periods>
<loaded_stations>
  <s Id="1" Load="0.30" />
  <s Id="2" Load="0.15" />
  <s Id="3" Load="0.10" />
</loaded_stations>
<messages_sizes>
  <m Length="1" Weight="1" Margin="1"/>
  ...
  <m Length="7" Weight="2" Margin="1"/>
  <m Length="8" Weight="8" Margin="2"/>
</messages_sizes>
</config>
```

Fig. 2. Excerpt of `body_config.xml`: an XML configuration file for the generation of message sets exchanged in a *body* network.

3.3 Parameter instantiation

One of the most important constraints imposed on an in-vehicle communication network is the maximum *load*, i.e., the average proportion of time the bus is busy transmitting messages. Since this value is application-specific, its specification is left to the user who can define in the configuration file a range that will be used by NETCARBENCH to choose a value from, whenever generating a new message set. The load range is specified by the line `<load Min="30" Max="35" />` in Figure 2. Once the load is set, NETCARBENCH will create as many messages as necessary to reach this load.

When creating a new message, NETCARBENCH first instantiates its *period* and *length* (i.e. length of the data payload in bytes), which are integer values whose definition interval is given in the configuration file. Since it cannot be assumed that the messages in real networks obey an uniform distribution for data lengths and transmission periods, user-defined probability distributions are given in the configuration file as frequency histograms. For instance, a period equals to 2 seconds will be allocated with a probability $5/\sum$Weight as specified by `<p Value="2000" Weight="5"`.

If sets of frames are to be generated, each frame must be assigned a *priority*. How priorities are allocated is specified along with the periods of the frames since, in practice, priorities tend to be assigned according to the Rate-Monotonic assignment: "the smaller the period, the higher the priority". As shown in Figure 2, the user can indeed associate to every possible period value a range of priorities (e.g., `PrioLowRange="1" PrioHighRange="200"`), from which NETCARBENCH will randomly select a value during the priority assignment step. Additionally, NETCARBENCH ensures the uniqueness of the frame priorities, and is able to handle potentially overlapping priority ranges that may have been specified for different periods in the configuration file.

Once the construction of the messages is completed, it remains to assign these messages to the different stations of the network. The number of nodes is constrained to remain within a specified range. For example, in Figure 2, one has `<ecu Min="15" Max ="20" />`. The messages are then distributed to the ECUs, in an uniform way if nothing else specified, or following a user-defined distribution. Indeed, it is typical in the *body* or *powertrain* network that one or more ECUs produce much more traffic than the others. For instance, as in Figure 2, it is possible to specify that node 1, 2 and 3 generate, respectively, 30,15 and 10 percent of the total load as specified by the lines:

```
<s Id="1" Load="0.30" />
<s Id="2" Load="0.15" />
<s Id="3" Load="0.10" />
```

Figure 3 shows an example XML file generated by NETCARBENCH describing a body network.

4. EVALUATION

The configuration file `body_config.xml` shown in Figure 2 models a typical *body network*. A set of 100 message sets were generated by NETCARBENCH from this configuration. In the following, we examine the quality of NETCARBENCH's output.

4.1 Distribution of the message lengths

The vertical bars in Figure 4 indicate the minimum, maximum and average values of the frequencies of the message lengths over the 100 generated message sets. In the graphic, these values should be compared with the dotted line, which connects the frequency (as defined by the weights) actually specified by the user in the element `<messages_sizes>` of the configuration file in Figure 2.

The weight of a length corresponds to its relative frequency in the message set, while the attribute `margin` in the configuration specifies the variability around this frequency. Precisely, if the user specifies for a

```
<msgset Busspeed="125"
  Name="set2.xml" Load="36.054%">
<ecu Name="Ecu_0">
  <frame Name="frame0" Priority="127"
    Period="50" Length="8" />
</ecu>
<ecu Name="Ecu_1">
  <frame Name="frame28" Priority="455"
    Period="100" Length="5" />
  ...
  <frame Name="frame56" Priority="942"
    Period="200" Length="8" />
</ecu>
...
<ecu Name="Ecu_19">
  <frame Name="frame67" Priority="210"
    Period="200" Length="1" />
  <frame Name="frame71" Priority="566"
    Period="200" Length="8" />
  <frame Name="frame76" Priority="614"
    Period="500" Length="7" />
  <frame Name="frame87" Priority="1284"
    Period="1000" Length="7" />
  <frame Name="frame91" Priority="394"
    Period="100" Length="8" />
</ecu>
</msgset>
```

Fig. 3. Excerpt of an XML file generated by NETCARBENCH modelling the message set in a *body* network.

length l a margin m and a weight w, the actual weight chosen by NETCARBENCH for the length l is then a random value in the range $R = [w - m, w + m]$. This approach was chosen to meet the requirements of letting the user describe precisely a specific configuration, while still allowing the generation of a sufficiently large number of different message sets complying with this specification. The strategy adopted here was not to consider the specification as a strict requirement but to make NETCARBENCH comply with it in the average case. The results in Figure 4 indeed reveal that NETCARBENCH sometimes selects weights that are outside this range (as shown by the extrema, especially the maximum for the length $l = 2$ which presents the largest overshot with respect to the upper bound of the specified range). However, for each data length, the average value observed coincides almost exactly with the weight initially specified by the user, which suggests that even on a relatively limited number of experiments, the message sets are representative of the user's specifications.

4.2 Assignment of the messages to the ECUs

In a similar manner as for the message lengths, we evaluate the assignment of the messages on the different stations of the network with the specification file in Figure 2. In the element `<loaded_stations>`, the user may, for every station i, define a *load* l_i that will directly determine the number of messages assigned to this ECU. As explained in Paragraph 3.3, we designed NETCARBENCH in such a way that it considers the stations one after the other and fills every ECU i

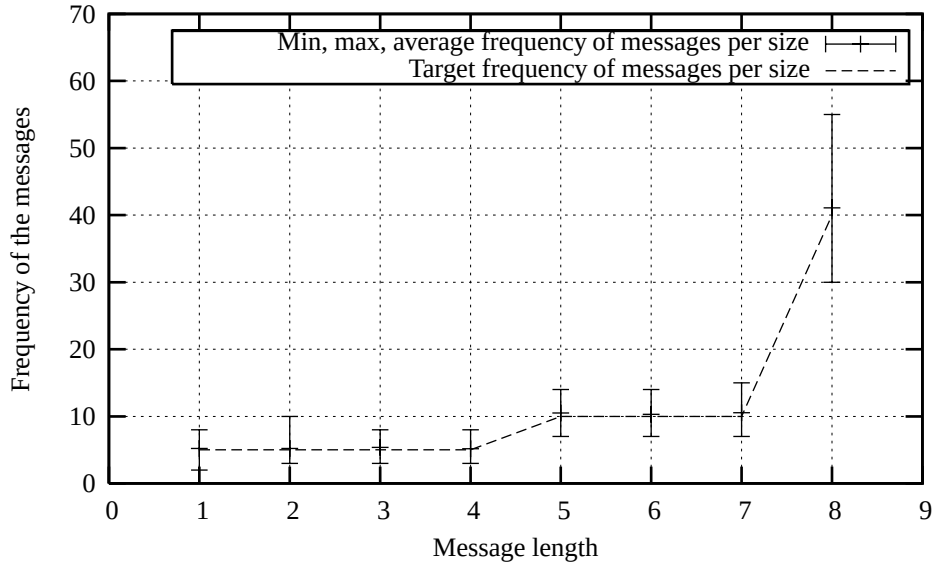

Fig. 4. Distribution of the message length (data payload in bytes) for 100 message sets generated by NETCARBENCH according to the configuration given in Figure 2.

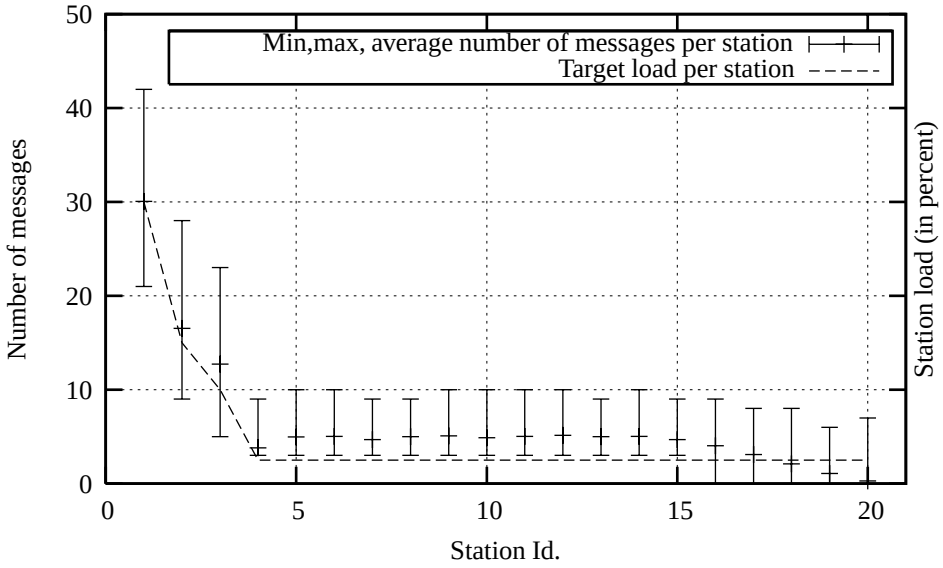

Fig. 5. Distribution of the network load (left-hand y-axis) and obtained number of messages per ECU (right-hand y-axis) in 100 message sets generated by NETCARBENCH according to the configuration given in Figure 2.

with messages until l_i^{max}, the first value of the load verifying $l_i \leq l_i^{max}$ is reached [4].

In Figure 5, the vertical bars on the graph correspond to the number of messages at each station for the 100 generated message sets. The dotted line represents the load for each station, which was either specified by the user in the configuration file, or calculated by default by NETCARBENCH as an average on the remaining unspecified stations. In our example, a load constraint was only specified for the first three ECUs, and the specification `<ecu Min="15" Max="20" />` leads to the generation of a network that contains at least 15 non-empty nodes, which explains that, starting from the fourth ECU, the dotted line remains horizontal.

The proximity between the average number of messages by stations and the dotted line, especially for the 3 first stations, shows that, logically, the number of messages and the load at a station are directly linked and a lower load implies a smaller number of messages assigned to this station. The large deviation on the number of messages (reflected by the lengths of the vertical bars for the 3 first stations) suggests that the actual local load assigned by NETCARBENCH to each station can take rather different values from one message set to another, which may be explained by the large range of possible data payload. Starting

[4] More precisely, NETCARBENCH decides to include or not include the last message with a probability 0.5, in order to avoid that the actual load is systematically above or below the user-defined threshold.

from the fourth station (i.e. for the stations whose load was not specified), one can observe a uniform distribution of the messages between the stations. The reduced load for the stations with identifiers greater than 16 can be explained by the fact that these stations do not always exist in the configuration generated by NETCARBENCH.

5. CONCLUSION

NETCARBENCH aims at improving the assessment, the understanding and the comparability of techniques and tools used in the design of in-vehicle communication networks. In particular, we hope that this benchmark might prove to be useful for the design of configuration algorithms and communication protocols.

One of the design objective of NETCARBENCH is to allow a fine-grained parameterization of the generated message sets, by means of XML-configuration files specifying the characteristics of the message sets and the variability thereof. Also, NETCARBENCH has been designed with the objective in mind that it remains suitable to evolutions in automotive communication, for instance an increased load or larger messages as it is possible in time-triggered networks (e.g., FlexRay).

An evaluation of the generated message sets revealed that the outputs comply with the user-defined specification. The XML output format adopted for both the input and output of NETCARBENCH eases the interoperability with other software tools.

In addition to the program itself, configuration files of typical *body* and *chassis* networks are provided. NETCARBENCH, and all the accompanying material, are licensed under the GNU General Public License version 2 (Free Software Foundation, Inc., n.d.).

REFERENCES

Braun, C., L. Havet and N. Navet (2007). NETCARBENCH user manual. Available with the source files at url `http://www.loria.fr/~nnavet/netcarbench/`.

Davis, R. I., A. Burns, R. J. Bril and J. J. Lukkien (2007). Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Syst.* **35**, 239–272.

Free Software Foundation, Inc. (n.d.). GNU GENERAL PUBLIC LICENSE, version 2, june 1991. Available at url `http://www.gnu.org/licenses/gpl.txt`.

Grenier, M., L. Havet and N. Navet (2008). Scheduling frames with offsets in automotive systems: a major performance boost. In: *Automotive Embedded Systems Handbook* (N. Navet and F. Simonot-Lion, Eds.). CRC Press/Taylor and Francis.

INRIA (2006). NETCAR-Analyzer: optimizing automotive networking.

ISO/DIS 11898 (1992). Road vehicles - interchange of digital information - controller area network (CAN) for high speed communication.

Navet, N., Y. Song and F. Simonot (2000). Worst-case deadline failure probability in real-time applications distributed over CAN (Controller Area Network). *Journal of Systems Architecture* **46**(7), 407–417.

Navet, N., Y. Song, F. Simonot-Lion and C. Wilwert (2005). Trends in automotive communication systems. *Proceedings of the IEEE* **93**(6), 1204–1223.

Norström, C., K. Sandström and M. Ahlmark (2000). Frame packing in real-time communication. In: *Proceedings of the Seventh International Conference on Real-Time Systems and Applications (RTCSA'00)*.

SAE (1994). Society of Automotive Engineer - SAE paper J2056/1 - Class C application requirementssociety of automotive engineer. *SAE Handbook* **2**, 23.272–23.366.

Saket, R. and N. Navet (2006). Frame packing algorithms for automotive applications. *Journal of Embedded Computing* **2**, 93–102.

SymtaVision (2007). SymTA/S Tool Suite. `http://www.symtavision.com`.

Tindell, K. and A. Burns (1994). Guaranteed message latencies for distributed safety-critical hard real-time control networks. In: *1st International CAN Conference Proceedings, Germany, September 1994*.

Appendix A. NETCARBENCH PSEUDOCODE

This Section provides a simplified description, in pseudo-code, of the algorithms used in NETCARBENCH to generate the message sets according to the configuration file.

```
PROC Netcarbench(ConfigFile,
  MsgSetNumber)
{
  SET OutputFiles:={};
  PARSE ConfigFile and SET
    LoadMin,LoadMax,EcuMin,
    EcuMax,EcuLoads,Bandwidth,
    IsSignal,Periods,Sizes,
    Priorities;

  FOR (i:=0;i<MsgSetNumber;i++)
  {
    SET OutputFiles[i]=
      GenerateNodes(LoadMin,
        LoadMax,EcuMin,EcuMax,
        EcuLoads,Bandwidth,
        IsSignal,Periods,Sizes,
        Priorities);
```

```
  }
  RETURN OutputFiles;
}

PROC GenerateNodes(LoadMin,
    LoadMax,EcuMin,EcuMax,
    EcuLoads,Bandwidth,IsSignal,
    Periods,Sizes,Priorities)
{
  SET MsgPriorities:=
    GetDistribution(
      Priorities);
  SET EcuNumber:=
    PickRandomInt(EcuMin,EcuMax);
  SET LoadLimit:=PickRandomInt(
    LoadMin,LoadMax);
  SET Messages:={};
  SET MsgPeriods:={};
  SET MsgSizes:={};
  SET Period:=0;
  SET Size:=0;
  SET Deadline:=0;
  SET LoadCurrent:=0;
  SET LoadTarget:=LoadLimit;

  WHILE EcuNumber>0
  {
    // Check if a load was
      specified for this Ecu.
    IF EcuNumber in EcuLoads
    {
      SET LoadTarget:=
        EcuLoads[EcuNumber];
    }
    // Add messages until the
      load limit is reached
    WHILE LoadCurrent<LoadTarget
    {
      // Generate if necessary
        the distribution of
        message periods
      IF MsgPeriods is empty
      {
        SET MsgPeriods:=
          GetDistribution(
            Periods);
      }
      SET Period:=
        PickRandomAndRemove(
          MsgPeriods);
      // Generate if necessary
        the distribution of
        message sizes
      IF MsgSizes is empty
      {
        SET MsgSizes:=
          GetDistribution(
            Sizes);
      }
      SET Size:=
        PickRandomAndRemove(
          MsgSizes);
      // Deadline on request
      SET Deadline:=Period;
      // Check that a unique
        priority is available
      IF MsgPriorities is empty
      {
        LOG("No available
          priority anymore");
        EXIT;
      }
      ELSE
      {
        SET Priority:=
          PickRandomAndRemove(
            MsgPriorities);
        // Create the message
        SET Message:=(Period,
          Deadline,Size,Priority);
        // Compute the additional
          load
        SET LoadCurrent+=
          GenerateLoad(Message,
            Bandwidth,IsSignal);
        ADD Message in
          Messages[EcuNumber];
      }
    }
    SET EcuNumber-=1;
  }
  RETURN
    GenerateOutputFile(Messages);
}
```