Automatic Parameter Identification in FlexRay based Automotive Communication Networks

Eric Armengaud and Andreas Steininger
Vienna University of Technology
{armengaud, steininger}@ecs.tuwien.ac.at

Martin Horauer
University of Applied Sciences Technikum Wien
horauer@technikum-wien.at

***Abstract*— Time-Triggered Architectures are being introduced in safety-critical automotive systems ("X-by-wire") to cope with the growing complexity and the high safety demands. One of their merits is to provide a static operation schedule, thus largely reducing the complexity of (otherwise input-dependent) execution flow. This, however, comes for the price of increased configuration complexity: The FlexRay protocol that implements the time-triggered paradigm on the communication level requires several tens of configuration parameters. The product of their possible settings spans a space of more than 10^{48} (theoretical) configurations. This does not represent a problem as long as the configuration is known and fault-free. However, in many cases – ranging from debugging over conformance testing to maintenance – an identification of the configuration is desirable, but turns out extremely burdensome. This paper presents a systematic approach that facilitates parameter identification in such complex systems. Experimental results illustrate the usefulness of our approach and explore its limitations.**

I. Introduction

Nowadays most automotive innovations stem from electronic systems. Typical cars are equipped with more than 50 electronic control units (ECUs), most of which being interconnected via broadcast networks. The use of these ECUs and especially their interoperation [1] allow the establishment of extended and improved functionality in comparison with stand-alone components (e.g., combining speed with steering information or combining multiple sensor information to a comprehensive picture of the car's surrounding). In this domain, the FlexRay protocol [2] has been defined by a large consortium of leading automotive and electronic OEMs to serve the needs of future distributed applications. Relying on the time-triggered paradigm [3] FlexRay addresses reliability and fault-tolerance aspects and provides the required bandwidth, while additional provisions also support event-triggered communication for an effective coupling of sensor/actuator systems with lower requirements.

FlexRay supports different network topologies – i.e. a passive bus, a star using star couplers and mixed topologies – and hosts two communication channels that can be optionally configured to operate in redundancy (fault tolerant systems). Media access control is based on recurring communication cycles consisting of (i) a static segment using a time division multiple access (TDMA) scheme and, (ii) an optional dynamic segment using a mini-slotting based scheme. In the static segment permission to access the bus is confined to predefined time windows (slots), leading to the advantage of a deterministic behavior. Distributed clock synchronization is the enabler for the TDMA scheme. Another very interesting property for automotive applications is the so called composability [4]. It allows to develop different subsystems independently, exactly simulate their final time behavior and subsequently to integrate them into the application. Generally the overall system architecture is defined by the OEM, which serves as an interface specification for the individual nodes implemented by the suppliers.

Drawbacks of time-triggered bus concepts are the lack of flexibility, sub-optimal resource usage, and the restrictive design process. In particular, all processes and their time specifications must be known in advance; otherwise, an efficient implementation is not possible. In contrast to traditional – event driven – protocols the static control flow necessitates a comprehensive definition of the temporal relations through additional parameters. In total, 74 parameters need to be configured, each with a suitable value within a predefined range. Now it becomes clear that for a system with more than 50 nodes – potentially from different suppliers – the configuration process can become a troublesome burden. There is, however, sufficient tool support for this task as long as the configuration is known and fault-free. Still in practice there is an appreciable number of cases where the configuration is not known or not trusted and shall hence be identified or verified. This paper details how we addressed these problems in the context of our research projects[1].

The paper is structured as follows: After a description of aims and requirements of our approach we will give a brief survey on related work. Our proposed method for parameter identification will then be introduced in Sec. IV. Next, Sec. V will be devoted to an experimental exploration of the approach. The benefits and limitations of the solution will be discussed in Sec. VI. The paper concludes with Sec. VII.

II. Motivation and Requirements

The deployment of configuration parameters is a small, yet essential part during the development process of automotive

[1]The STEACS-project (Systematic Test of Embedded Automotive Communication Systems) received support from the Austrian "FIT-IT embedded systems" initiative under grant 807146. Furthermore, the DECS-project (Design methods for Embedded Control Systems) received support form the Austrian FHplus research initiative under grant 811414.

applications. Most of the time the intended parameter settings for a cluster and its respective nodes are at hand and stored in suitable documentation files. Even when this documentation should be missing, one might argue that the configuration can be read back out of the configured nodes. This, however, requires direct (intrusive!) access to every node in the system or dedicated operating system support for reading out the parameters via an (operational) network. Moreover, an existing configuration is not necessarily correct and trustworthy (consider the case of debugging, e.g.). So in practice, there is an appreciable number of situations where an identification of configuration parameters during or after system deployment might be beneficial. The following list categorizes these use-cases:

Parameter check: Following the deployment of an application to all the nodes in the system a system-wide end-to-end test is desirable. For this purpose the cluster can be put into operation and the configuration be identified and checked against the application specification. Unlike a mere read-back of the configuration this kind of check also covers the correct execution of the respective settings by the communication controller. Furthermore, a parameter identification of this kind can turn out very useful as a quick maintenance check, too.

Parameter measurement: Measuring and checking configured parameters can be used to identify minor mismatches in the network setup or borderline cases with respect to tolerances that do not completely inhibit communication but cause intermittent problems.

Parameter characterization: Characterization implies the determination of the tolerances exhibited by every single parameter. This characterization in turn allows for a check whether the behavior is still within the specification – an issue that is of interest for a conformance test. In addition parameter characterization may be useful for maintenance as well.

Parameter surveillance and logging: Monitoring the parameters online over time allows assessing and identifying sporadic parameter deviations (an approach is described in [5]) and the gradual change of parameters over time due to ageing effects. By using statistics and trend analysis one will be able to identify weak components before they fail (active maintenance [6]). Furthermore, logging of selected parameters is a very concise form of information representation useful for routine checks and post-mortem analysis.

The FlexRay protocol specification [7] lists 74 parameters that can be used to adapt specific properties of the bus traffic to the needs of the application. For every parameter p_i a valid range is specified, or, to be more precise, a finite and discrete set $\mathbb{P}_i^*$ of *permissible values*. The network controller provides dedicated registers to which the desired parameter settings can be written. We call a complete and consistent image of such register entries the *node configuration*, and the compilation of the configurations of all nodes within the network the *network configuration*. All parameters whose values are part of the node configuration form the *parameter set* of a node.

A key problem for the desired parameter identification is the large parameter space spanned by this parameter set and the associated permissible values. Tab. I gives a rough overview, without being able to go into details: Parameters related to the frame syntax and bus schedule have been compiled into groups (I) and (II), with group (II) representing parameters relevant in context with the dynamic segment only. Group (III) exemplifies some parameters used in context with clock synchronization, start-up, or configuration of properties that are local to a node only. The focus of our paper will be on group (I), since the parameters contained herein are most relevant for the above mentioned use-cases (consensus on syntax and medium access). Still, the product of the parameter ranges of these parameters spans a space of more than 10^{24} possible configurations for one node.

From an implementation point of view parameter identification should be applicable in various different phases of a product lifecycle. Hence, requirements are

(R1) to come along with a remote tester node whose only connection with the system under test is via the FlexRay communication network

(R2) an identification that is completely transparent for the system under test, which means
  (a) no change in the network structure
  (b) no change in the bus access behavior / schedule
  (c) no change in the node (HW, SW) or application (value and timing)

This essentially means that no explicit support from the target system is available – a challenging aim.

III. Related Work

Monitoring of distributed embedded systems is frequently used for debugging, logging and other analysis on the data level; see [8], [9] for some recent advances. Commercial tools for the analysis and diagnosis of the most popular automotive or real-time communication protocols are available. Examples are CANalyzer[2] for the CAN protocol, the TTP monitoring-card or -node[3] for the TTP protocol, or the BusDoctor[4], and the CANalyzer[5] expansion for the FlexRay protocol. All these solutions, however, only enable the monitoring of the bus traffic on top of the data link or higher layer (plus some error flags). For the identification of the configuration parameters of a communication protocol this needs to be complemented by bus monitoring and dedicated measurements on lower abstraction levels, as well as an appropriate data analysis. Therefore, none of those tools currently provide means to analyze or identify (part of) the configuration.

Some implementations related to the identification and discovery of basic configuration parameters have been patented and published, see e.g. [10], [11]. These approaches address different – typically event triggered – communication protocols with a significantly smaller parameter space, such that an unstructured search is applicable. Knowledge discovery

[2]http://www.canalyzer.com/
[3]http://www.tttech.com/
[4]http://www.decomsys.com/
[5]http://www.vector-informatik.com/

TABLE I

A SUBSET OF FLEXRAY CONFIGURATION PARAMETERS WITH THEIR VALUE SPACE AND SCOPE

class	FlexRay Parameter	$\lvert\mathbb{P}_i^*\rvert$	Scope	class	FlexRay Parameter	$\lvert\mathbb{P}_i^*\rvert$	Scope
I	gdBit	3	frame syntax	II	gdDynamicSlotIdlePhase	3	bus schedule
	gdTSSTransmitter	11	frame syntax		gMaxPayloadLengthDynamic	127	bus schedule
	gPayloadLengthStatic	64	frame syntax		gdMinislot	62	bus schedule
	gdCycle	15989	bus schedule		gdMinislotActionPointOffset	31	bus schedule
	gdStaticSlot	2043	bus schedule	III	gdMacrotick	255	clock-sync
	gNumberOfStaticSlots	1022	bus schedule		pOffsetCorrectionOut	7680	clock-sync
	gdNIT	772	bus schedule		pRateCorrectionOut	1922	clock-sync
	gdActionPointOffset	31	bus schedule		gOffsetCorrectionStart	15988	clock-sync
	gdSymbolWindow	87	bus schedule		pDelayCompensation[A]	127	topology
	gNumberOfMinislot	7994	bus schedule		...	...	

approaches have been used in the network domain for topology discovery, see e.g. [12]. The main drawback of the latter approach is that it employs a challenge/response scheme, which is highly intrusive.

IV. PROPOSED CONCEPT

Given the requirements from Sec. II we assume the target system to be a set of distributed computing nodes that are connected over a communication network based on the FlexRay protocol. Herein, our aim is to determine the network configuration of this given, running target system. Therefore, we attach a dedicated monitoring node to the bus and use a remote host to perform the data interpretation, filtering, and parameter identification steps, cf. Fig. 1.

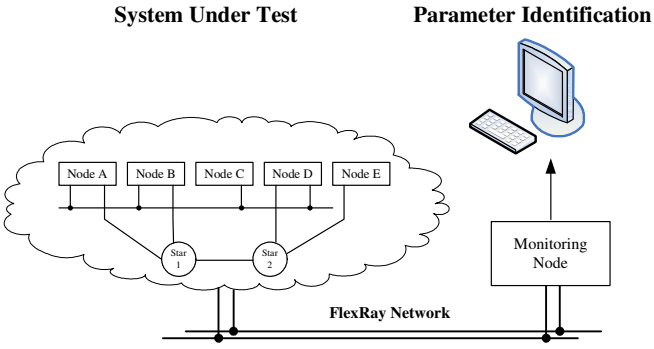

Fig. 1. System Setup

Similar to knowledge discovery [13] in the artificial intelligence domain, our approach comprises four major steps:
Structuring: Given the huge parameter space it is evident that some kind of "divide and conquer" is required to cope with the complexity of the problem. We propose a fine-grained layer structure that decomposes the communication service into a number of atomic *mechanisms*, each characterized by one or a few parameters, cf. [14].
Accessibility: Our layer concept only makes sense, if input and output of all mechanisms can actually be accessed, such that the individual service provided by a mechanism can be directly observed. Since a standard node does not provide sufficient support here, we have devised a dedicated monitoring node [15] with extended capabilities for our purpose.
Determination: Once the task has been reduced to determining one single parameter (or a few parameters) of an isolated mechanism, we use two complementary strategies, namely measurement and tuning.
Interpretation: In most cases the determination does not directly yield the desired parameter value. We have to apply statistics in order to suppress undesired effects, and – since there are many cases where a parameter is not directly visible on the bus – we have to transform the measurement result to the quantity of interest.

A. Structuring the Parameter Space

It is the task of the network controller's receive service to transform a serial signal received from the FlexRay bus via the physical line interface into the enclosed payload data that can finally be used within the application context. This involves a lot of activities to assemble, strip and check the received information (frame). We decompose this complex process into a set of individual sub-services S_i that are provided by so-called mechanisms M_i. Ideally, a mechanism has one single information input, one single information output and a status output. Its operation is described by a simple model in a generic way, and one or several parameters (and constants) can be used to characterize it. Both, the generic operation model (including the constants) as well as the set of applicable parameters and their permissible values can be extracted from the protocol specification. Let us consider the NRZ decoder as an example. This mechanism receives the glitch-filtered serial bit stream as its input. Its task is to remove the synchronization elements between consecutive bytes. It recognizes these by counting the bit sequence from the beginning of a frame and removing the last two bits from every group of ten bits – in this case a constant rule that cannot be configured. At its output it provides a decoded bit stream plus a status whether a coding error occurred (e.g. missing or erroneous byte start sequence).

The same decomposition approach is applicable for the transmit path, too. In fact our tester can only observe a nodes' transmission behavior. Given that transmit channel and receive channel use the same configuration (anything else would not make sense), we can project the parameters determined for the transmit channel to the receive channel as well.

Our approach is somewhat similar to the OSI layer model, however, finer-grained. In [14] we have illustrated the com-

plete model for the FlexRay communication services. The following key property of this model shall be highlighted that will become important later on: Mechanisms can be hierarchically ordered in *levels*, such that a high-level mechanism M_k builds upon the services provided by the lower-layer mechanisms $M_{k-1} \dots M_1$. Therefore the service of mechanism M_k does not suffer from an erroneous behavior or incorrect configuration of a higher-layer mechanism M_{k+i}, while it does suffer from an error of a lower layer mechanism M_{k-i}. This property allows us to identify the parameters one by one, starting with the lowest level mechanism and successively increasing the level. Thereby every step builds upon the identification results of the previous one(s).
This dramatically reduces the search efforts from a multi-dimensional space to one single dimension at a time. In some rare cases mechanisms comprise two parameters that are mutually dependent on each other, such that the search is two-dimensional – still a significant saving.

B. Accessibility

The proposed decomposition is only meaningful, if we can indeed evaluate the mechanisms individually. For this purpose we need access to every mechanism M_i's input, output and status information. We have designed a dedicated monitoring node that basically works like a standard communication controller, but is additionally capable of tracing information at all required observation points. During the development of this prototype it turned out that the decomposition into mechanisms reflects the hardware implementation quite well (in fact the typical HDL-design of a communication controller is nothing else than a properly orchestrated collection of modules implementing mechanisms) such that it was pretty easy to provide access to the desired information in the first place. The difficult points were (a) the high bandwidth required for tracing and transferring the detail information collected on lower levels to the host and (b) maintaining the temporal correlation between events collected on different levels. We used buffering and time-stamping to overcome these problems, for details see [15]. In fact, for economic reasons our implementation does not provide access to each and every level. Instead, we decided to derive the information on some carefully selected levels L_k from the trace of the lower level L_{k-1} by emulating the service of the enclosed mechanism M_{k-1}. The price is a degraded decomposition of the involved parameters.

C. Parameter Determination Strategies

The above measures put us in the position to have input stream, output stream, status and model of a given mechanism M_i available, and the remaining task is to determine the respective parameter such that these pieces of information fit together and produce the correct (fault-free) output.

Tuning: One approach here is to "tune" the parameter value within the allowed range until there is a match. In practice this involves the following steps:

For the determination of parameter p_i associated with mechanism M_i

- S1: identify all Parameters $p_1 \dots p_{i-1}$ and configure Mechanisms $M_1 \dots M_{i-1}$ accordingly (As pointed out in Sec. IV-A, the parameters are identified one by one starting with the lowest mechanism M_1).
- S2: select a starting value for p_i
- S3: configure M_i with this value
- S4: observe the status output of M_i to decide whether the chosen setting matches the observed bus traffic:
  - a: if the status is "error" then select another valid setting for p_i and proceed with step S3
  - b: if the status is "ok" the chosen value is correct and the parameter was successfully identified

The strategy for selecting a starting value and for stepping through the elements of $\mathbb{P}_i^*$ can be optimized (i.e. the most probable settings first) by using protocol know-how. In this way the identification process can be speeded up.

In some cases more than one parameter setting may be valid (see below). This means that every result from step S4b is stored as successful and the process is continued with step S3 until the complete list has been processed.

Measurement: In a general sense protocol parameters are used to shape the appearance of the bus traffic. This, in turn, means that every parameter setting is visible in the bus traffic in some way. Therefore it is a very natural approach for parameter determination to measure some characteristics of the bus traffic. This may be as simple as directly determining a parameter such as the baud rate or the cycle length. In other cases, however, parameters are not directly visible and must be indirectly determined. Examples are the network idle time or macrotick length. Finally, it may also occur that a parameter cannot be determined at all, since the mechanism associated with it is simply not exercised during the observed portion of the communication. An example for this is a boundary value (min, max) that is unlikely to be exceeded during an observation.

This shows a fundamental limitation of the identification approach: Hardly any communication scenario exploits all features of the protocol, and hence communication is often possible even if not all parameters are correct or congruent. Possible reasons are that the communication schedule is not dense, dynamic slots are not utilized or boundary conditions are not reached. Depending on whether the respective protocol feature will be utilized in the given setting later on such a mismatch may or may not become visible. There is no way for the identification process to make a decision between (seemingly) equivalent configurations.

As an example let us consider the (frequent) case that the last slots within a communication cycle are unused. With respect to such an observation it makes no difference where the start of the Network Idle Time (NIT) is assumed, which makes the identification of the parameters *gdNIT* and *gNumberOfMinislots* (or *gNumberOfStaticSlots* when no dynamic part is used) ambiguous. Integrating a node with parameters

that are not congruent with the configuration of the other nodes in this respect will cause no problems as long as the last slots remain unused.

The decision between tuning and measurement is very much dependent on the application purpose and implementation details. In principle, both methods are equivalently applicable for every parameter. In practice, however, it turned out that the approaches are rather complementary: Tuning puts lower requirements on the monitoring hardware (basically, only the error status must be accessible) and proved very efficient for parameters that have a relatively small set of permissible values. Measurement, on the other hand, is beneficial when the parameter in question can be directly observed and access to the data inputs and outputs is easy. In practice, however, these data streams can quite easily be derived from a trace performed on a low abstraction level and an appropriate software emulation for the "by-passed" mechanisms. In fact a bit-level trace along with an associated time stamp for the frame start allowed us to identify, check and strip all relevant information concerning transmission start sequence, frame header, CRC and payload. On the other hand, the determination of the bit length by means of the measurement algorithm would require us to trace and evaluate the oversampled bit stream – this task can be easier performed by using the tuning approach.

D. Interpretation and Filtering

It is not surprising that the exploitation of protocol-specific know-how can significantly increase the efficiency of the identification process. Two examples for this in the above description are the proper choice of a starting point for tuning and the interpretation leading from a measurement result to a parameter value. There is, however, much more potential in an intelligent interpretation of the plain results and observations: Not all combinations of valid parameter settings, e.g., yield a valid and reasonable configuration. In some cases it is therefore possible to restrict the range of higher level parameters based on the knowledge of lower level parameters and protocol restrictions. The knowledge, for instance, that the offset correction must start within the NIT and finish before the end of the NIT allows us to derive boundary values for the parameter *gOffsetCorrectionStart* once the NIT has been identified. Another example is the macrotick length: The knowledge that this parameter must be a common divisor of the static slot length, the network idle time, the symbol window length, the minislot length, and others, usually allows to narrow down $\mathbb{P}_i^*$ for this parameter to a few choices.

The meaningful presentation of the determined parameter values to the user is another important issue. In the ideal case an unambiguous and complete set of precisely determined values – one for each parameter – can be presented as the result of the identification process. In practice, however, this turned out to be rarely the case. This is due to the following reasons:

1) As already argued above, not all relevant effects may have shown up during the observation interval. As a consequence some parameters are difficult to identify; often it will only be possible to narrow their range.
2) Since every node operates with its own local crystal clock oscillator (microtick), activities are essentially uncorrelated from a microscopic view (while a globally agreed time base is established on a macroscopic level). This inevitably leads to a measurement jitter, at least on lower levels.
3) As a result of drift effects (temperature drift of the crystals, e.g.) the communication properties of the observed system are not completely static. A prominent example for this is the drift of the global system clock; for FlexRay this is allowed to vary within the specified limits of $1500ppm$. This drift directly causes a variation of the cycle length and other parameters.
4) It cannot be taken for granted that the observed communication is completely fault-free. Therefore, it is important to establish some means for suppressing outliers, since these would lead to conflicting results otherwise.

With respect to issues 3 and 4 it may as well be the purpose of the identification process to detect such outliers and assess their frequency, or it may be the intention to assess the parameter variations. As outlined in Sec. IV-C this depends on the application context of the identification.

In any case it is generally not sufficient to perform a single pass of the identification process and compute the results. In general a longer observation yields a more precise result (and in the ideal case the observation is continuous). This raises the question of when to finish the observation process, and it appears reasonable to give the control to the user and provide him with information related to the trustworthiness of the results. The latter may be a residual interval for a parameter and/or statistic qualifiers like the number of samples or the standard deviation.

V. Experimental Validation

In order to demonstrate the practical usefulness of our approach and explore its limitations we performed an experimental investigation on our prototype implementation. In particular we wanted to study how its efficiency depends on network configuration and bus traffic density (bus load), and finally evaluate its robustness against transient faults. The setup for our experiments consisted of a COTS FlexRay cluster, our dedicated monitoring node and a host computer for parameter identification as illustrated in Fig 1. For a given scenario, a complete identification took approximately 30 seconds and each parameter was identified more than 1000 times.

A. Metrics

- **Identification status:** For every parameter p_i a set of permissible values is given in the specification (as a range with defined step size, for example). This initial value set $\mathbb{P}_i^*$ is hence known a priori, and it is the purpose of the identification process to identify the precise parameter value c_i for a given cluster or at least reduce $\mathbb{P}_i^*$ by ruling out elements, such that the cardinality $|\mathbb{P}_i|$ of the

set of remaining possible values is as small as possible. In our experiments the correct setting $c_i \in \mathbb{P}_i^*$ for every parameter p_i is of course known and can hence be used for checking the correctness of the identification result. In this context the result of the identification of a given parameter can be classified as

  - "successful": $\mathbb{P}_i = c_i$ (the correct value was exactly identified)
  - "correct": $|\mathbb{P}_i| > 1$ and $c_i \in \mathbb{P}_i \subset \mathbb{P}_i^*$ (set was reduced & contains the correct value)
  - "not detected": $\mathbb{P}_i \equiv \mathbb{P}_i^*$ (the set could not be reduced at all)
  - "faulty": $c_i \notin \mathbb{P}_i$ (the set determined does not contain the correct value)

- **(Local) Efficiency:** We want the efficiency metric to express – on a scale between 0% and 100% – how far the identification process has aided us in precisely determining the actual parameter value c_i of a parameter p_i or at least in reducing the initial value set $\mathbb{P}_i^*$. Logically, an efficiency of 100% implies that the parameter has been "successfully" identified, while 0% denotes that no reduction of $\mathbb{P}_i^*$ has been accomplished ("not detected"). Given that (a) many parameters exhibit a large set of permissible values ($|\mathbb{P}_i^*| > 1000$, e.g.) and that we (b) strive for a significant reduction of this set, it seems appropriate to use a logarithmic scale for expressing the amount of this reduction. This leads to the following formal definition of the (local) efficiency E_i for the identification of a parameter p_i : $E_i = 1 - \lg(|\mathbb{P}_i|)/\lg(|\mathbb{P}_i^*|)$ [%]. Notice that E_i expresses the relative improvement in the exponent of $|\mathbb{P}_i|$ with respect to the initial set $|\mathbb{P}_i^*|$. Consequently for the same absolute improvement E_i tends to become larger when $|\mathbb{P}_i|$ approaches 1.
- **Global Efficiency:** In analogy to the local efficiency, the global efficiency is calculated as $GE = 1 - \sum \lg(|\mathbb{P}_i|)/\sum \lg(|\mathbb{P}_i^*|)$. Global efficiency denotes how far the aim of identifying the parameter vector $(c_0, c_1, c_2, \ldots c_n)$ describing the configuration of the system under consideration has been accomplished: In this sense a value of 100% indicates that the configuration has been "successfully" identified while 0% mean that no reduction at all has been attained.

B. Experimental Efficiency Assessment

Tab. II shows the identification results for scenario CF3 (see Tab. III) with 70% bus traffic density. It can be seen that 7 out of 10 parameters could be successfully identified ($E_i = 100\%$), which is a remarkable achievement in any case. Since the application did not use dynamic messages, only trivial results could be identified for the related parameters, which are therefore excluded from the calculation of the global efficiency.

A similar case is *gdSymbolWindow* that describes the length of the symbol window. Since no symbol was observed during this experiment (20 seconds of bus traffic), it was not possible to identify the length. Unlike the case of dynamic frames, we regarded this as a "natural" effect in an identification process rather than a deficiency of our setup and therefore considered the unfavorable result ($E_i = 0\%$) in the calculation of GE.

TABLE II
EXPERIMENTAL RESULTS FOR SCENARIO CF3, 70% TRAFFIC DENSITY

Parameter	$\mathbb{P}_i^*$	$\mathbb{P}_i$	E_i
gdBit	3	1	100.00%
gdCycle	15989	1	100.00%
gdStaticSlot	2043	1	100.00%
gNumberOfStaticSlot	1022	1	100.00%
gPayloadLengthStatic	64	1	100.00%
gdNIT	772	767	0.10%
gdTSSTransmitter	11	1	100.00%
gdActionPointOffset	31	24	7.45%
gdSymbolWindow	87	87	0.00%
gNumberOfMinislot	7994	1	100.00%
gdDynamicSlotIdlePhase	-	-	-
gMaxPayloadLengthDynamic	-	-	-
gdMinislot	-	-	-
gdMinislotActionPointOffset	-	-	-
Total Space / Global Efficiency	1.17E+24	1.60E+06	74.21%

The parameter *gdActionPointOffset* describes the frame offset within the slot which is mandatory to increase the robustness against clock drift. Since this parameter largely suffers from local oscillator drifts, it cannot be precisely identified without having access to the nodes' local view of time. By appropriately relating information on actual slot length and frame length $E_i = 7.45\%$ could at least be achieved.

As already discussed above *gdNIT* is not directly visible on the network, and its identification efficiency largely depends on the network traffic. Our bad result of ($E_i = 0.1\%$) is partly due to an unfavorable traffic constellation, but still an improvement by a more sophisticated interpretation strategy is conceivable.

Overall, the applicable search space could be reduced from $1.17{*}10^{24}$ to $1.60{*}10^{6}$, i.e. by almost 18 orders of magnitude, yielding a global efficiency of $GE = 74.22\%$. Our correct implementation of the identification tools was confirmed by the fact that no faulty detection occurred.

C. Impact of the Bus Traffic Density

The bus traffic density (or bus load) – expressed as the proportion of bus activity within a communication cycle – was varied between the values 5%, 30%, 70% and >90% for configuration CF3 (see Tab. III). The frames were positioned randomly within the communication cycle, but always in accordance with the FlexRay specification.

Fig. 2a illustrates the dependence of GE on the network traffic density at the example of configuration CF3. It can be observed that an increase of the network traffic density from 5% to 100% improves GE from 62% to 74% (i.e. by an absolute factor of 50). A closer analysis revealed that this improvement only affected the parameters *gNumberOfStaticSlots* and *gdNIT*. This indicates that increased traffic tends to contribute to a clearer perception of the segment boundaries. More precisely, high network traffic increases the chances for observing "valuable" frames that are positioned right at the boundaries of interest. While this is true for the general case,

TABLE III
NETWORK CONFIGURATIONS USED FOR THE EVALUATION

FlexRay Parameter	CF1	CF2	CF3	CF4	CF5
gdBit [Mbps]	10	10	5	10	2,5
gdCycle [μs]	3000	5582	10780	180	7401
gdStaticSlot [μs]	20	16	90	16	175
gNumberOfStaticSlot [slot]	142	118	118	2	2
gPayloadLengthStatic [byte]	3	16	16	1	12
gdNIT [μs]	151	145	151	139	392
gdTSSTransmitter [bit]	8	11	11	11	15
gdActionPointOffset [μs]	1	1	1	1	1
gdSymbolWindow [μs]	9	9	9	9	9
gNumberOfMinislots [slots]	0	0	0	0	353
gdDynamicSlotIdlePhase [bit]	1	1	1	1	1
gMaxPayloadLengthDyn. [byte]	0	0	0	0	16
gdMinislot [μs]	3	3	3	3	15
gdMinislotActionPointOff. [μs]	1	1	1	1	1

there may be a more favorable schedule that yields higher GE than a less favorable one for the same network traffic density.

Interestingly GE does not noticeably change when reducing the network traffic from 30% to 5%. This indicates that there is a significant share of parameters whose local efficiency is (at least largely) independent from the network traffic. A similar saturation effect can be observed in our results for an increase from 70% to 100%: Even a dense traffic cannot compensate for local oscillator drifts and the lack of symbols.

D. Impact of the Configuration

Five different network configurations (CF1 ... CF5) were used to evaluate the dependence of identification quality on diverse settings. These respective settings are summarized in Tab. III (only the parameters that are within our scope are listed).

During this campaign, the configurations CF1 ... CF5 were compared, all with 5% bus traffic. Fig. 2b shows GE plotted for these different configurations. It can be observed that the variation of GE remains within 1%, which is quite surprising considering the large differences among the configurations. Similar to Sec. V-C the choice of the configuration only affected the local efficiency for *gdActionPointOffset*, *gNumberOfStaticSlots* and *gdNIT*.

E. Robustness Evaluation

As already outlined in Sec. IV-D we do not rely on a single measurement but apply statistics to multiple determination results such that outliers can be suppressed (or identified, if desired). In our experiment setup we discarded the lowest 10% and the highest 10% of the results from the determination step (commonly known as "quantile") to get rid of transient fault effects. In order to experimentally validate this robustness we superposed the following faults to an originally fault free bus traffic:

- Transient syntax errors: The fault rate in this experiment was $10^{-3}, 10^{-2}$ and 10^{-1} single bit flips per frame, which resulted in 18, 180 and 1808 faults during the 5 second observation period.
- Transient parameter deviations: The fault rate was $10^{-3}, 10^{-2}$ and 10^{-1} cycle length extensions per cycle, yielding 1, 10 and 95 faults during the 5 second observation period.

Fig. 2c compares the global efficiency GE attained under the above faults with the fault free case. In all fault scenarios GE remains at the original level of 62.14%, which proves the expected robustness to faulty bus traffic. In addition, each deviation was properly indicated in the report files of the identification process.

VI. BENEFITS AND LIMITATIONS

As our experimental results show, it is indeed possible to attain reasonable results even under the tight industrial constraints stated in Sec. II. In practice the presented experimental results are by far sufficient for a "plug-and-play" integration of a passive ("listen only") node into a running cluster. The same is true for an active node (i.e. one that may also send) provided that the bus traffic observed during the identification contained all relevant protocol features (for our experiments this would mean that there are indeed no dynamic frames) and the global schedule was set up accordingly, beforehand. While safety critical applications will probably not adopt these plug-and-play features, they may benefit e.g. from the parameter surveillance enabled by our approach.

Some problems, however, still remain unsolved:

- As already outlined above, the distinction between different configurations that are equivalent with respect to the observed bus traffic is not possible. This seems to remain a fundamental limitation unless our monitoring node is allowed to apply stimuli. Therefore we are planning to complement our parameter identification with the option to actively stimulate the network by means of a dedicated active tester node. While this extension obviously sacrifices the transparency with respect to the network traffic, it allows a more comprehensive identification. In addition we have to convey information on the reception status of the stimulus from the node back to the tester. A way to accomplish this without modifications at the node's side is currently being investigated in a separate research project.[6]
- By listening to the bus traffic we have a quite direct view on the nodes' transmit paths. Since it does not make sense to use divergent parameter settings for transmit path and receive path (and since this is usually not supported by the controller hardware), it is reasonable to project the identification results to the receive path as well. However, in the context with more advanced applications of the parameter identification (characterization of boundary values or path-specific constants, e.g.) this projection may become insufficient and some kind of stimulation paired with a loopback from the top of a node's receive path to the tester again becomes inevitable.
- In the current concept the monitoring node is viewed as an ideal reference. All observed faults and inaccuracies

[6]See http://www.ecs.tuwien.ac.at/mitarbeiter/armengaud/extract.html for details on the ExTraCT project

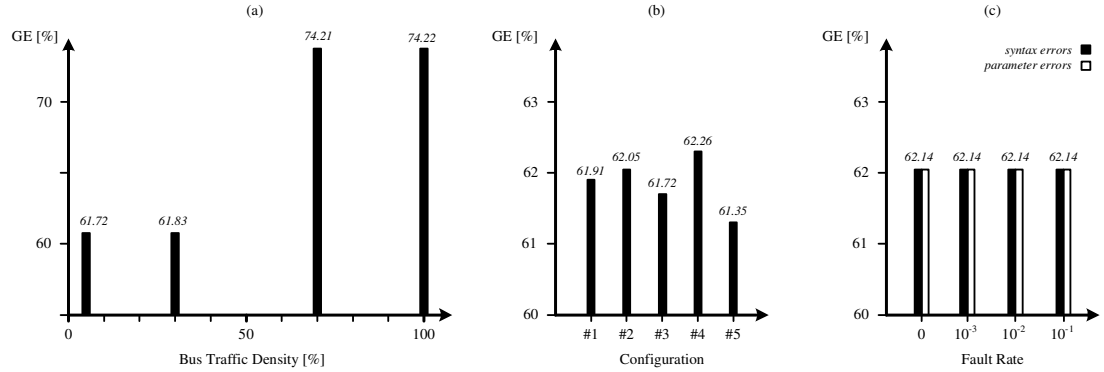

Fig. 2. Identification quality vs. bus traffic density (a); Identification quality vs. cluster configuration (b); Identification quality vs. fault rate (c)

are attributed to the target system. While this supposition may be legitimate for a node that is periodically checked and calibrated, it definitely needs to be reconsidered for an embedded solution used, e.g., in context with parameter surveillance.

- The parameter identification tool used in our experiments does not cover parameters for start-up and clock synchronization. Active stimulation and the option to restart the nodes on demand would be required for this purpose.

The basic concepts of our approach can be applied to other communication protocols as well, although a different protocol specification with different parameters and value sets obviously implies different rules for the determination step and the interpretation, particularly for event driven protocols.

VII. Conclusion and Future Prospects

We have proposed a strategy for automatic identification of the network configuration in a FlexRay network. The approach employs a remote node that passively listens to the network traffic from which all relevant information is extracted. This makes the approach perfectly transparent. The strategy comprises four steps, namely (i) structuring the complex protocol service into simple mechanisms, (ii) providing access to all relevant data within a dedicated monitoring node, (iii) using the tuning approach and – as a complement – measurement of traffic properties to determine the parameter values, and (iv) applying protocol know-how and statistics to narrow down the remaining parameter space to suppress undesired effects.

Our experimental evaluations on a prototype implementation of a monitoring node and the associated software tools have yielded very encouraging results. At the same time several limitations have shown up, some of which can be overcome by improving the concept and implementation, while others appear to be fundamental. Our next steps will be directed towards including an option for stimulus generation into the tools and implementing a transparent way for observing the reception status of a node. We have identified numerous relevant application domains, where a tool for parameter identification can be very useful, and even in its current prototype implementation our approach already provides very versatile support from debugging prototypes in a lab to an efficient maintenance in a garage.

References

[1] D. Marsh, "Network Protocols Compete for Highway Supremacy," *EDN Europe*, pp. 26–38, June 2003.

[2] R. Mores, G. Hay, R. Belschner, J. Berwanger, C. Ebner, S. Fluhrer, E. Fuchs, B. Hedenetz, W. Kuffner, A. Krüger, P. Lohrmann, D. Millinger, M. Peller, J. Ruh, A. Schedl, and M. Sprachmann, "FlexRay - The Communication System for Advanced Automotive Control Systems." *Society of Automotive Engineers (SAE) 2001 World Congres*, March 2001.

[3] H. Kopetz and G. Bauer, "The Time-Triggered Architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112 – 126, Jan. 2003.

[4] H. Kopetz and R. Obermaisser, "Temporal Composability," *Computing & Control Engineering Journal*, vol. 13, no. 4, pp. 156–162, Aug. 2002.

[5] E. Armengaud, F. Rothensteiner, A. Steininger, R. Pallierer, M. Horauer, and M. Zauner, "A Structured Approach for the Systematic Test of Embedded Automotive Communication Systems," in *IEEE International Test Conference (ITC2005)*, Nov. 2005, pp. 1–8.

[6] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[7] "Flexray Communications Systems – Protocol Specification Version 2.0," FlexRay Consortium, 2004.

[8] I. Smaili, "Real-time monitoring for the time-triggered architecture," Ph.D. dissertation, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2004.

[9] H. Thane, "Monitoring, Testing and Debugging of Distributed Real-Time Systems," Ph.D. dissertation, Mälardalen Real-Time Research Centre (MRTC), Department of Computer Engineering, Mälardalen University (MDH), 2000.

[10] U. Walter and V. Lolov, "Baudrate Detection in Serial Data Transimission," Intl. Patent: WO 99-26384, 1999.

[11] L. Wilhelmsson, "Method and Apparatus for Iterative Parameter Estimation," US. Patent Application: US 2002-0067782, 2002.

[12] S. Motegi, K. Yoshihara, and H. Horiuchi, "Service Discovery for Wireless Ad Hoc Networks," *5th International Symposium on Wireless Personal Multimedia Communications*, vol. 1, pp. 232–236, October 2002.

[13] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery: An overview." in *Advances in Knowledge Discovery and Data Mining*, 1996, pp. 1–34.

[14] E. Armengaud, A. Steininger, M. Horauer, and R. Pallierer, "A Layer Model for the Systematic Test of Time-Triggered Automotive Communication Systems," 5^{th} *IEEE International Workshop on Factory Communication Systems*, pp. 275–283, September 2004.

[15] E. Armengaud, A. Steininger, M. Horauer, R. Pallierer, and H. Friedl, "A Monitoring Concept for an Automotive Distributed Network - The FlexRay Example," 7^{th} *IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems (DDECS'04)*, pp. 173–178, April 2004.