Pushing the Limits of Remote Online Diagnosis in FlexRay-based Networks

Eric Armengaud[1,2], Andreas Steininger[1]

[1] Vienna University of Technology, Embedded Computing Systems Group E182-2,
Treitlstr. 3, A-1040 Vienna, {armengaud, steininger}@ecs.tuwien.ac.at

[2] DECOMSYS GmbH, Stumpergasse 48/28, A-1060 Vienna

Abstract

In an automotive distributed embedded system direct access to individual nodes for testing is often impossible during field operation. Remote test access over the communication network – albeit suboptimal from a testability point of view – is sometimes the only viable solution from an economic point of view. At the example of the FlexRay communication protocol we investigate which diagnostic conclusions with respect to the communication services can be drawn from remotely monitoring the normal bus traffic. In particular we discuss how the the clock synchronization service can be exploited for online diagnosis. We further propose a novel approach for transparent online testing that utilizes the clock synchronization services to loop back information on a node's reception status to the tester.

1. Introduction

Electronics is the key innovation driver in the automotive domain. Analysts estimate that more than 90 percent of all automotive innovations now stem from electronic systems [1]. In modern upper class vehicles several hundreds of signals are exchanged among up to 70 electronic control units (ECUs), which turns nowadays cars into highly distributed control systems [2]. The use of these ECUs and especially their interoperation allow the establishment of extended and improved functionality in comparison with traditional stand-alone components (e.g., combining speed with steering information or combining multiple sensor values to a comprehensive picture of the car's surrounding). These distributed automotive control systems are developed under stringent cost constraints. In order to reduce costs for cabling and ease future improvements, the distributed ECUs are interconnected by a serial broadcast network.

For advanced applications in future automotive control systems ("X-by-wire systems") an industrial consortium of leading automotive and electronic manufacturers has established a protocol termed FlexRay [3]. Relying on the time-triggered paradigm [4], FlexRay combines reliability and fault-tolerance with a bandwidth suitable for a communication backbone, while at the same time providing sufficient flexibility for the coupling of sensor/actuator systems.

For the development, operation, service, and maintenance of such complex distributed systems powerful facilities for monitoring, testing and diagnosis are required; see [5] for some related discussions and approaches. Furthermore, means for fault injection are necessary to test the error handling and fault tolerance properties and assess the reliability of the system under inspection [6]. Considering the distributed nature of these systems, the related concepts for testing and fault injection must go beyond the component borderlines. In contrast to (the traditional understanding of) the "divide and conquer" strategy it is vital to test the assembled system rather than its isolated components. While the concept of *composability* [7] provides solid theoretical foundations for efficient module integration, validation of the assembled system is still required.

It is then the aim of this work[1] to increase the reliability of the assembled system; more specifically to provide a guarantee, that once the system has started it will either provide correct services or that service deviation will be detected and diagnosed. The resulting diagnostic information can be further used for logging or reporting the system status, or as base for targeted recovery actions.

From a practical point of view an ideal tester for such a distributed embedded system has the following properties:

(P1): It needs to be connected to the communication network only, since other test points are often impractical to access.

(P2): It is completely non-intrusive with respect to the system behavior (value and time domain). This not only eliminates the "probe effect" but also allows testing to be performed online.

(P3): It does not require any supportive provisions or architectural changes in the system under test. This last

[1]The ExTraCT-project received support from the Austrian FIT-IT Embedded Systems initiative, funded by the Austrian Ministry for Traffic, Innovation and Technology (BMVIT) and managed by the Austrian Research Promotion Agency (FFG) under grant 810834.

point is important to keep the test approach generic, i.e. not specific to a platform or application.

Obviously these restrictions tend to compromise the attainable diagnostic capabilities of such a tester.

In this paper we will investigate, which diagnostic conclusions can still be drawn from merely listening to the network traffic, and we will identify the associated limitations. Our specific contribution will be to highlight the role of the clock synchronization service in this context and to propose a novel approach that is currently being further pursued in our ExTraCT project[2].

The paper is organized as follows: First, we present the system architecture in Section 2 and discuss testability issues involved with the chosen test approach in Section 3. Next, in Section 4, we discuss several options for remote monitoring and investigate how much diagnostic information can be gained. In Section 5 we present and analyze a novel approach that utilizes the clock synchronization service for transparent online testing. Finally, we conclude our paper with some directions for future enhancements in Section 6.

2. System Architecture

The time-triggered architecture [4] has been introduced to ease the development of distributed real-time systems. Its concept is based on a globally synchronized sparse time [8] to which both the external events and the system progression are synchronized. Furthermore, process execution is deterministically controlled by a static, predefined schedule, thus reducing the number of possible program paths to exactly one [9]. Amongst other things this approach provides the major advantages (a) to solve the temporal ordering problem, and (b) to minimize the system complexity in providing a deterministic and a-priori known program execution path.

Examples of Time-Triggered communication protocols are TTP [10] for aircrafts or FlexRay [3] for automobiles. These protocols implement a Time Division Multiple Access (TDMA) scheme based on a-priori defined time windows ("communication slots") which are uniquely assigned to the nodes for message transmission within a periodic communication cycle. Main characteristics are (a) bounded communication delays independent of the traffic load, and (b) the accurate interface definition both in the time and value domain, thus enabling temporal composability and easing system development and enhancement. Fig. 1 illustrates a small distributed FlexRay system.

Central problems for application development and maintenance of such distributed systems are debugging and testing of the assembled system. Assuming that nodes have been tested in isolation already, methods to detect errors during operation are required. One practical approach that is applicable for the different phases of a product lifecycle is to add a dedicated tester node to the distributed

[2]http://www.ecs.tuwien.ac.at/∼armengaud for further information.

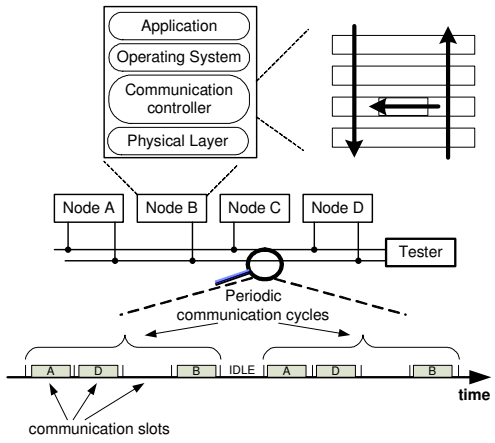


Figure 1. An example for a considered system architecture

system and perform a remote test as shown in Fig. 1. Since this is usually a single dedicated node, its cost is relatively uncritical for the overall system cost. In contrast, changes at the nodes in the system under test shall be avoided. Notice that – according to property (P1) stated in Section 1 – in this approach the tester only requires access to the communication network at an arbitrary point.

The elimination of the need to directly access signals on any node makes the solution extremely cheap and elegant. At the same time, however two conditions must be met: (i) To start with, a communication between the tester and all other nodes over the network must be possible. This is not self-evident in case of a severe node error or configuration error. However, given that we assume every node to be already tested in isolation this will be an extremely rare case that we do not further consider in the following. (ii) All node-internal information required by the tester for diagnosis must be available on the communication network. (Later on in this paper we will se how far this is actually the case and what must be done to complete information that might be missing). This approach can be compared with *hardware monitoring* [5], where the monitoring hardware is directly located at the serial network instead of the local parallel bus of the node under test.

The nodes within the system typically have a regular architecture as pictured in Fig. 1. The scope of this work is set to the communication services, and therefore to the correct inter-operation of all the communication controllers involved. A communication controller typically consists of (a) a transmit path to write the data to the network, (b) a receive path to receive the data and (c) a medium access control to avoid collision during transmission and check correct reception. In our case a clock synchronization mechanism associated with a scheduler plays this role; see [11] for a fine-grained layer model of the FlexRay protocol. In context with our further analysis it is interesting to note here, that – as usual in protocols – the actual message contents are protected by numerous

mechanisms (framing, CRC, etc.). These protections are enforced upon transmission and checked upon reception. As a consequence the receive path maintains a number of error flags (each corresponding to one protection mechanism) that form an "error status" – a property not present in the transmit path. Obviously this error status will become important for diagnosis later on.

It should be mentioned here that in the scope of this paper we only consider single faults and assume the tester to be fault-free; otherwise, we cannot distinguish between a receiver fault of the tester and a transmission fault of a node in the system under test. Since we have already decided to allow higher cost and efforts for the tester, an extensive self-test can be performed before putting the tester into operation. Since the function of the tester is not mandatory for the (primary) service of the system, the tester may even go off-line periodically and repeat this self-test.

3. Classification of our Approach

We consider our approach basically a remote online monitoring method, with an extension towards remote online testing as presented in Section 5. To clarify this statement we will briefly survey the related terminology in the following.

3.1. Remote Testing/Monitoring

The quality of a test result (in terms of test speed and coverage) attainable with given efforts strongly depends on the testability of the system under test. According to [12] the two key aspects for testability are controllability and observability. Controllability is an indicator of how easy it is to bring the system or node under test to a given state. Observability is a measure of how easy it is to observe certain activities of the system or node under test. Both measures are largely determined by the accessibility of the system under test. In short, good accessibility leads to good testability.

In our remote approach we want to avoid dedicated test points at the individual nodes. Instead we want to restrict the access of the tester to the network to match property (P3) desired in Section 1. Indeed, the communication network is an excellent access point for a tester due to its unique role for the communication among the distributed nodes. Still, however, all information that is local to a node, such as its status, is not available at the communication network. In summary the key problem with our remote testing approach is that the serial communication network provides very limited options to observe and control the system. Therefore we have to expect low testability and as a consequence low test coverage.

3.2. Monitoring vs. Testing

The normal test procedure is a stimulus/response measurement: Well chosen test vectors are applied to the system under test and the response is observed and compared to a known good reference. This approach by its nature causes an interference with a potentially ongoing operation of the system under test, which contradicts property (P2) from Section 1.

However, under certain conditions the normal system operation can be considered a sufficient set of test vectors, and hence the requirement to explicitly apply stimuli can be relaxed. It is sufficient to observe the ongoing system operation instead and check whether it meets the expectations. For this purpose some kind of rule checking is usually employed, e.g. [13]. The main question when using this monitoring approach is whether all relevant system functions are exercised during normal system operation within the observation time. Usually, without explicit stimulation some exceptional states (e.g., emergency handling) are very likely not to be entered and the associated resources hence not to be exercised and tested. This limited test coverage may cause problems in context with latent errors as outlined in [14]. The major advantage of monitoring is that – since no active stimulation occurs – it is not necessarily intrusive since it can be performed in a transparent way. In addition, the sequence of test vectors comes for free and is representative for the actual application. Our first strategy will therefore be to rely on monitoring of the network traffic.

3.3. Online vs. Off-line Testing

In the classical sense the test is a very specific mode of operation of the system under test: The system is taken off-line, a well chosen comprehensive set of test vectors is sequentially applied to the system under test to step through a sequence of states and thoroughly execute all relevant functions. The resulting system behavior is monitored and compared to a known good reference [15, 16]. The theory for this off-line test approach is well developed and the test usually achieves good coverage, since having complete control over the system facilitates good testability. Online testing, in contrast, is performed while the system under test is still providing its service. The main advantages of online testing are:

- Online testing may be continuously ongoing, which allows very fast detection of errors – as compared with off-line tests that are usually performed in long intervals. Especially, online testing is a means to prevent fault accumulation. While the test activities might jeopardize the system reliability, it has been shown in [14] that the negative impact of fault accumulation may be even worse.
- The continuous operation of the online test allows collecting a comprehensive and representative test record. This in turn facilitates statistical assessment and projections, e.g., the identification of "weak" components that are liable to fail soon ("active maintenance [17]").
- An online test is by its definition non-intrusive with respect to the service delivery. This means that it

does not compete with other system tasks for resources and does not modify the node's status in an observable way. As a consequence, it has no probe effect.

Especially the last issue is of central interest for our purpose: The inter-operation of the nodes in a distributed system often suffers from very subtle effects, and this is where a probe effect is most troublesome. Therefore we are willing to pay the price for online testing, namely the high efforts required for keeping the test transparent for the ongoing system operation. In particular this means that (i) the test must not change any application data, and (ii) in a real-time environment the test must not influence the timing behavior of the application. This is an extremely tough requirement, given that the usual test procedure comprises a stimulation by the tester. In combination with our approach, however, online monitoring suggests itself. Later on, in Section 5 we will also propose a method for transparent stimulation of the system under test.

4. Remote Online Monitoring

4.1. Direct Diagnosis

Being restricted to remote access we have to carefully exploit every nuance of information we are able to observe on the communication network. And indeed, the FlexRay specification facilitates a lot of checks to be performed on a message:

- **Time domain:** The static transmission schedule allows to determine in advance, when exactly a node has to transmit a message. Not only sending at the wrong instant (incorrect temporal alignment) but also the omission of a message can be identified as a failure.
- **Value domain/syntax:** The syntactic correctness of a message can be checked with respect to coding, CRC, header integrity, etc.
- **Value domain/semantics:** Given that the tester is provided with application specific knowledge, several assertions can be applied to the semantic correctness of a message, such as boundary checks, plausibility tests, model-based checks, etc.

  This last option, however, is out of the scope of a generic communication service test.

For our remote test setup this means that the tester can indeed passively listen to the traffic on the communication network and perform the above checks to judge the integrity of the messages. As soon as an abnormal behavior is detected, the tester can – thanks to the static schedule – easily project this error to the producing node (except for severe errors in the time domain). On the other hand if the tester receives correct messages from a remote node, some important conclusions can be drawn:

- After having received at least one correct message from each node that participates in the communication we can conclude that there must be an intact link between the tester and any other node. For passive bus topologies this also means that an operational link between any two nodes exists as well.
- The basic services of the transmit path of that node must work properly, otherwise we would have encountered syntactic errors.
- The basic clock synchronization services must work properly, otherwise the frame would not be properly aligned.
- Depending on application and assertions we may be able to conclude that the application has produced correct output messages based on correctly received input messages.

This is already a very useful amount of diagnostic information, and in context with a clever interpretation the status of the communication subsystem can already be characterized quite well. Let us briefly summarize the coverage that we have attained so far:

Transmit path: We can detect permanent and transient errors in the transmit path safely, since the output of this path is directly observable. Moreover, due to the time-triggered nature of the communication, the services within the transmit path are periodically exercised. In the transmitter there are no exceptional paths and no mechanisms for error detection, whose status must be observed. The correspondence between the service layers of sender and receiver pointed out in [11] allows us to use the error symptoms at the tester's receive path for diagnosing the actual error in the sender's transmit path.

Receive path: The situation is much more difficult here: The receive path processes information from the network and provides it to the application, which means that we can observe its input but not its output. As outlined in Section 2 the receive path additionally maintains an error status that is in fact much more important for diagnosis than the actual message contents (these can be derived from the observed network traffic anyway). Obviously what we are lacking here is some mechanism to loop back this reception status information to the network where it can be observed. We may use assertions on application level to serve this purpose. However, even if these are available, fault tolerance features of the application tend to mask the loss of single messages.

Clock synchronization service: The function of the clock synchronization service can, in principle, either be judged by the alignment of the transmitted frame, or by

the absence of scheduled transmissions: Without a synchronization the node will go to a silent mode. An identification of the failing node is easily possible, further diagnostic information, however, not available.

In conclusion we may be quite satisfied with the coverage of the transmit path. What we are definitely lacking is a way to observe the status of the receive path with our remote tester. Therefore we will survey several options to loop back this information to the communication network, and in particular investigate the clock synchronization service in this context.

4.2. Options for Improvement

There are two main approaches to improve observability within the node and loop back the node's receiver status. *Hardware monitoring*, on one side, requires additional hardware to monitor the node-internal electrical signals [18, 19]. Such methods are non-intrusive because they do not modify the node behavior, neither in the time nor value domain; however their cost is high since additional hardware is required, and they are impractical for the desired remote test. On the other side, *software monitoring* methods [20, 21] do not require additional hardware but compete with the application for internal resources to perform their operation and return the node's status. They are therefore intrusive.

Software monitoring can be implemented at different abstraction levels. One popular option is the operating system level. This choice not only relieves the application programmer from the testing issues, it also represents a solution that can remain unchanged for all types of application running on top of the operating system. A main advantage of this solution is that message related information and node status are readily accessible, although in a reduced form. The operating system, however, does not have access to application specific information.

Another possible approach is to implement the loop-back as application task, which allows access to application status and, via system calls, to node status as well. Since communication protocol and operating system usually relieve the application from communication specific details, these may become difficult to access. Another drawback of this approach is that it requires the application programmer to explicitly consider protocol related testing issues, which is expensive and error prone. If the application changes, it will very likely become necessary to change the loop-back issues as well.

The intrusiveness of software monitoring methods can be hidden in the context of time-triggered architectures [4] by carefully aligning the additional tasks with CPU idle times in the original schedule. The method is then non-intrusive since the node's behavior with respect to the application is not modified. Such approaches, however, are not generic, since they work only when there is (a) unused CPU time available for processing and (b) bandwidth available to transmit the node's status.

In addition, the lack of standardization makes this solution difficult to generalize. Dependencies on underlying operating systems or application might decrease the reusability and increase the development costs. An industry-wide initiative to standardize automotive system architecture (AUTOSAR) has been presented by Heinecke et al. [22]. While in this initiative an entire work package is dedicated to test and integration, there is no discussion about the means to reach these goals.

However, implementing a task in software to explicitly return (periodically) each node's status is not the only way to gain remote observability of the node's receiver status. In time-triggered communication protocols the clock synchronization service is making use of correctly received frames to schedule its own transmission. It is the aim of the next section to explore this implicit loop-back.

4.3. Exploiting the Loop-back via the Clock Synchronization Service

The basic service provided by the clock synchronization algorithm in time-triggered architectures is a system-wide, globally synchronized (sparse) local time base maintained by a so-called Macrotick count $CP(t)$. Among other things this time base forms a fundamental prerequisite to maintain the TDMA scheme. To this end it is used to assign every incoming *sync* (synchronization) message m_i a time stamp $CP_i^{TS} = CP(t_i)$ according to its arrival time t_i. The static bus access schedule allows to predict a reference value CP_i^{ref} for this time stamp (corresponding to the point in time when the message is expected to arrive according to the local alignment of the schedule). After timestamping, the incoming message propagates through higher layers where several checks are performed (e.g. frame alignment, CRC) and the header is analyzed. If the message passes these checks and the header indicates a "sync message", then the message is qualified for use in the synchronization algorithm. In this case the deviation between receive time stamp and reference value of this message m_i, formally $\Delta_i = CP_i^{TS} - CP_i^{ref}$, is stored. At the end of every cluster cycle the deviation values $\Delta_i \forall i \in 1...n$ (with $n \leq 15$) of all n valid sync messages received are subjected to the *fault tolerant midpoint* (FTM) algorithm, see [23]. According to this algorithm when *n* values were received the *k* highest and the *k* lowest deviations are discarded and the truncated midpoint of the rest is calculated. Herein, the parameter *k* is derived from the overall number of received clock synchronization packets *n*, see [23]. The resulting value is finally used to correct the local clock such that the desired global precision can be maintained.

Based on the local clock and the static bus schedule every node determines the points in time within a cluster cycle when it is supposed to send. Should its local time (or its understanding of the global schedule) deviate, the node will incorrectly align its message with the global schedule. This in turn can be perceived by the tester either as a shifted message, or as a truncated message or a message

omission when an optional bus guardian prevents the node from sending outside its pre-assigned transmission slot.

In short this means that every sync message that is received correctly by a node N_x contributes to N_x's clock correction, which, in turn, is visible at the alignment of the messages sent by N_x. So apparently the clock synchronization service forms the desired loop-back of the receive path error status to the communication network. This is illustrated in Fig. 2.

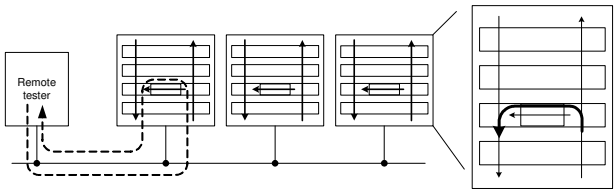


Figure 2. Loop-back via the Clock Synchronization Service

To exploit this loop-back let us summarize several important properties of the clock synchronization service from the description above:

1. The algorithm only considers sync messages. All other messages (and their loss or failure of reception, respectively) have no influence on the local clock synchronization.

2. For a sync message to be considered by the algorithm it must have passed semantic checks in the value domain (like CRC check) and a frame alignment check in the time domain, which are quite selective. With a faulty receiver it is very unlikely for a message to pass these tests.

3. The algorithm can tolerate the loss of sync messages without problems, and only requires two sync frames per cluster cycle to maintain its synchronization.

4. If too few sync messages are received, then the node changes its mode to "listen only" for some cycles and switches off after a pre-configured time-out, unless it can successfully receive further sync frames.

From (2) we can conclude that permanent errors in the receive path will eventually lead to loss of synchrony and hence be detected by the remote tester – according to (4) as a message omission. Furthermore, it follows from (1) and (3) that the loss of single messages due to transient faults in the node under test's receive path is not detectable by the remote tester, even if a sync message is affected. According to (3) a transient fault must affect a sufficient number of sync messages during several succeeding rounds to become detectable. Hence, transient faults are very unlikely to lead to a loss of synchrony – which is good news from the point of view of robustness, but inhibits their remote detection.

From the proper function of the clock synchronization services, on the other hand, we can conclude that the clock synchronization algorithm must have received sync messages recently. Otherwise the node would have eventually lost synchronization. This further implies with high probability that

- the basic services of the receive path must work at least up to the frame check level (this is where the decision is made whether to consider a sync message for clock correction) and that

- there must have been at least one successful reception during the previous communication cycle, otherwise the node would have turned silent.

Beyond this "go/nogo" check on whether the messages sent by the node are properly aligned within the schedule, it is conceivable to additionally perform a more subtle test on the position of the node's messages within the allowed time window. This position, however, is influenced by many effects and the variation resulting from the loss of one sync message is very small. Therefore, even if the tester can resolve the frame position precisely enough to recognize such variations, it is extremely difficult to distinguish the desired effect from variations caused by other sources.

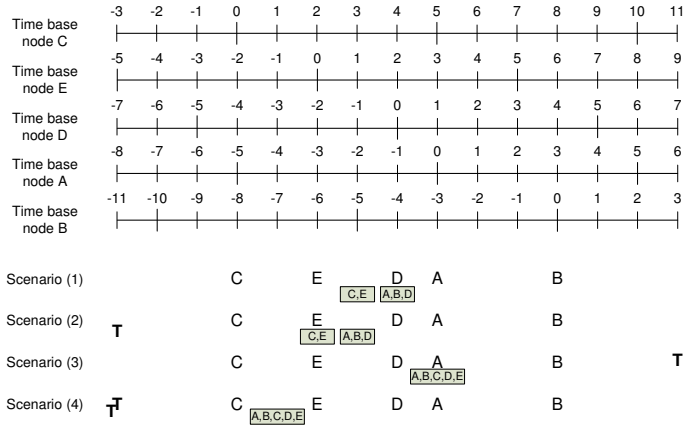
5. A Novel Approach for Transparent Online Testing

In the previous section we have identified the loop-back formed by the clock synchronization service and employed it for a rough go/nogo check of a node's receive path. Still, the diagnostic capabilities of this approach are far from satisfactory, and we still have no means to test the error detection capabilities in the receive path. At the same time we already know that there is more information hidden in the frame position than we currently exploit. So let us go one step further.

5.1. Basic Idea

We already know that it makes a difference for the result of the clock synchronization algorithm whether a sync message has been correctly received by a node or not, and that this difference becomes visible in the position of the frame transmitted by this node. Unfortunately – for our purpose – the individual local clocks are tightly synchronized, and potential outliers are suppressed by the FTM algorithm. So the loss of one sync message won't make much difference in practice. But what if our tester actively transmits messages that do have a measurable influence on the frame position if they are correctly received? Then the lack of such a reaction will in turn allow us to conclude that the test message has not been correctly received.

All the tester has to do for this purpose is to send one regular message ("stimulus message") within one communication round that (*i*) is a sync message, (*ii*) exhibits a



FTM Algorithm

Node \ Scenario	(1)	(2)	(3)	(4)
A	-1	-2	0	-4
B	-4	-5	-3	-7
C	+3	+2	+5	+1
D	0	-1	+1	-3
E	+1	0	+3	-1

Figure 3. Actively influencing the clock synchronization

sufficient deviation from the "normal" time base to cause a visible impact on the clock synchronization algorithm (i.e. represents a "stimulus"), but (*iii*) is still considered as valid. Condition (*i*) is easy to fulfil. Conditions (*ii*) and (*iii*) can, in principle, be met by positioning the stimulus message to an extreme location within the allowed time window. Fig. 3 shows an illustrative numerical example:

The upper left part of the figure represents the mutual alignment of the internal time bases from the different nodes. The lower part shows how the individual messages transmissions fit into this time base. Notice that a message is always sent at time '0' according to the sender's time base. For example, the message transmitted by node D has an offset of -1 for node A and an offset of +4 for node C. Four scenarios are pictured here: First, as reference, without tester node (scenario 1), then with a single tester frame with negative offset (scenario 2) and positive offset (scenario 3) and finally with a tester sending two frames with an important negative offset (scenario 4). The results of the clock correction algorithm (FTM) are summarized in the top right of the figure for the different scenarios.

Let assume now that node D in the system under test perceives the following individual deviations from its local time base {-4, -2, 0, +1, +4} measured in "time units", which might be ticks of the local clock. If the tester did not interfere, the FTM algorithm would discard the highest (+4) and the lowest (-4) deviation and compute a (truncated) average over the remaining interval borders, which yields (0). By sending a stimulus message with a deviation of (-7) or (+7), respectively, the tester can cause node D to move the result of its FTM to offsets (-1) or (1), respectively, in this example.

Notice that the stimulus message was in fact not considered in the midpoint, but rather discarded as an outlier. By occupying the outlier position, however, it caused the original outlier to be considered, which yielded a visible effect. In practice, however, true outliers will be rare – in the fault free case the clock synchronization will keep all local clocks within tight bounds. Therefore pushing a value from the outlier position into the average will not necessarily cause a noticeable effect. Obviously the solution is to let the tester occupy all outlier positions plus one or more values with stimulus messages (Scenario 4 of Fig. 3) to actively influence the result. This is relatively easy to achieve, since the FlexRay protocol [23] defines a maximum of 15 synchronization frames per communication cycle and a maximum of four outliers (the two highest and the two lowest) to discard. Hence, the generation of 3 synchronization frames is enough to directly influence the result of the FTM. If more than the 15 allowed synchronization frames are sent within a cycle (e.g. when the application already sends 15 sync frames and the tester additionally three), then the first 15 are taken into account and an error is signaled to the hosts. Consequently, this method only requires three empty slots (out of the 2047 possible identifiers) prior to the 13^{th} synchronization frame.

As compared with the mere monitoring approach discussed in Section 4 the proposed careful control of the clock synchronization services explicitly exercises the loop-back path and enables observability of the receiver status, which provides important advantages:

- For a node that adapts to the above changes properly we can conclude that it must have properly received the tester's stimulus message(s). Otherwise we can conclude that the receiver has encountered an error upon reception of the stimuli.
- By exercising the ability of the clock synchronization service to adapt to changes in the network timing we attain a better coverage of the clock synchronization services.

It is important to notice here, that this approach is still completely transparent for the application service. The tester's stimulus messages simply remain unused by the application, and even if they should fail to be received (e.g., due to an overly aggressive choice of their displacement) this makes no difference. Moreover, we still do not require any changes on the system under test, we only exploit the existence of the clock synchronization services

in a transparent way. The only resources we consume is the bandwidth for three frames. One might argue here that the "unaligned" tester messages tend to unduely exhaust the fault-tolerance of the clock synchronization. We are currently investigating this point more closely. In general, however, the negative impact of fault accumulation, that can be eliminated by the proposed on-line testing (as shown in [14]) constitutes a much more severe dependability threat.

The requirement on the tester for this type of test is the ability to send messages according to a very fine-grained time scale. In order to observe the reaction of the global time base, the tester must be able to sufficiently resolve the position of a message within the assigned time slot.

5.2. Taking Control of the Clock Synchronization Service for the Loop-back

The next step in exploiting the clock synchronization service for testing is to have the tester generate sync frames in such a way that that it essentially takes control of the remote nodes' clock correction mechanisms over time and forces them to follow a non conventional but still correct time pattern. Fig. 4 illustrates two experiments in which this approach is used to test the maximum offset and rate correction supported by the nodes. Both graphics illustrate how the cycle length (measured as the time interval between two consecutive frames with same identifier) evolves over time. In both cases the tester starts generating a conventional bus traffic. Next the system under test is started and synchronizes to this traffic. After communication cycle 1000 the actual test starts. In the left figure the tester gradually increases the cycle length. It can be observed that the nodes under test follow this artificial behavior until the tolerance limits of their clock synchronization services are exceeded (near cycle 1900) and they loose synchronization. In the right figure the situation is similar. Here the variation of the cycle length is kept below the tolerance limits, but the rate of change (slope of the curve) is gradually increased. Again the system under test looses its synchronization as soon as the respective limits are exceeded.

In both experiments the tolerance of the clock synchronization service has been overstressed, and as a result the nodes under test turn silent and communication breaks down. While such a test is still valuable for an off-line characterization of cluster parameters (e.g., conformance testing), it is not suitable any more for the transparent online testing we are aiming at.

However, we need not go that far. Depending on the perceived fault scenario a FlexRay node can exhibit different reactions: It may (1) discard the faulty message and continue providing its services, (2) turn into a listen only mode or (3) shut down after a timeout. What we experienced at the end of our previous experiments (Fig. 4) was scenario (2) then (3). In context with our transparent testing scenario (1) seems much more interesting: We can again use our tester to force the time synchronization to a "non-natural" state – in contrast to the experiments shown in 4, however, we do not go beyond the capabilities of the clock synchronization services. This still allows us to distinguish whether or not the stimulus messages have been received. By modifying these stimulus messages (insertion of faults, e.g., for a possible strategy see [24]) we can exercise exceptional paths and remotely trigger the node's error detection mechanisms. While this remains completely transparent for the application (the stimulus messages are not considered by the application), we can exploit the loop-back to observe the reaction of the receive path to the stimulus.

The following experiment illustrates this claim: We are considering a (simplified) cluster consisting of two nodes that transmit one frame each. They are configured for a communication cycle length of 3000μs and assumed to be fault free during the whole experiment.

In addition a tester generates two stimulus messages with the aim of driving the communication cycle length down to 2998μs. All messages are fault free but for communication cycles 1930 to 1939 during which both stimulus messages exhibit header CRC errors (a total of 10 frames for each channel were modified).

We first started up the tester alone. After 1000 cycles we allowed the nodes under test to integrate into the ongoing network traffic. After further 1000 cycles the tester stopped generating messages and the nodes under test were free running. If our above hypotheses are true we can expect that:

- The test cluster will be able to integrate on the network traffic generated by the tester, even if the cycle length slightly differs from the one configured
- The test cluster will stay synchronized and thus provide its normal services
- The test cluster will produce an observable but specification compliant reaction to the fault injected and still deliver its services without any alteration.

Fig. 5 pictures the cycle length of the four frames over time. As expected, the test cluster integrated to the existing network traffic (shortly before communication cycle 1000), and stayed synchronous although the cycle length was shortened by 2μs. When the stimuli messages ceased, the transmission period of the cluster returned close to the configured value of 3000μs . Indeed the faulty frames sent between the cycles 1930 and 1940 lead to a reaction of the nodes under test but did not crash the communication services. The errors contained in these messages were correctly detected (thus providing evidence for the proper operation of the CRC checker) and discarded from the internal clock correction computation. During these 10 cycles, the cluster started to move back towards the configured 3000μs transmission period until the tester messages were fault-free again.

This simple experiment illustrated the transparency of the approach for the application. Indeed, the additional

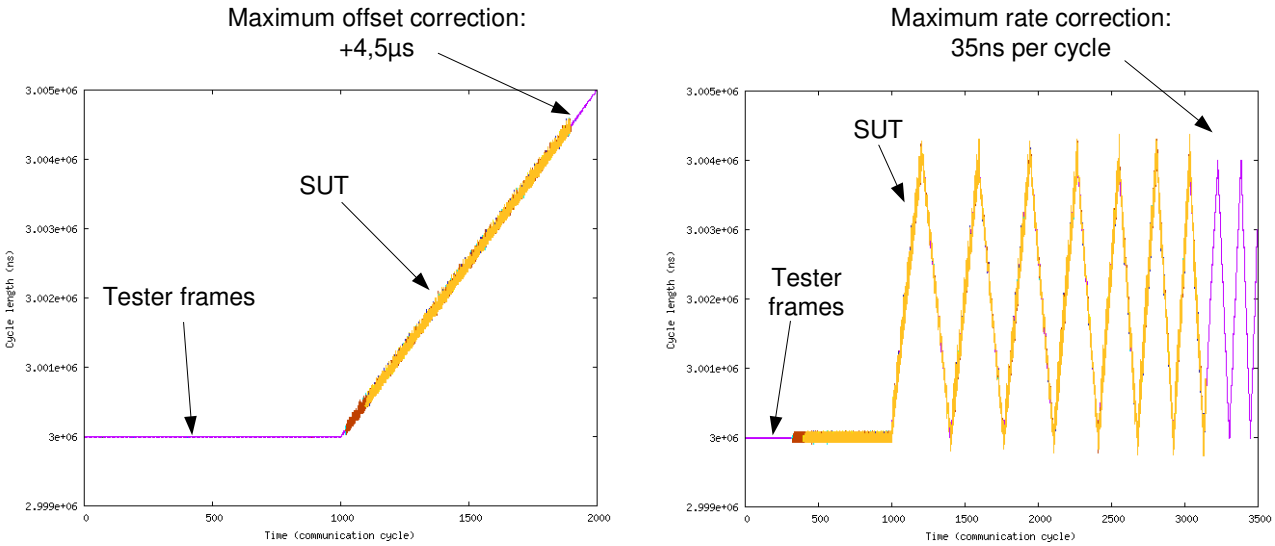


Figure 4. Driving the clock synchronization service to the limit

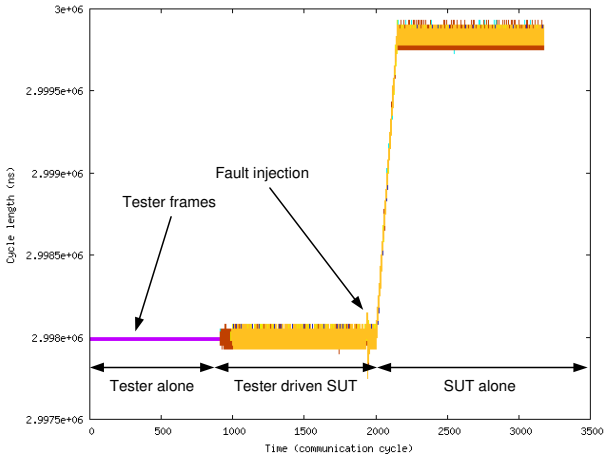


Figure 5. Fault injection experiment

frames generated by the tester were ignored by the application, and the two frames representing the application-related communication were transmitted without any disturbance. This behavior results from the constructive integration attribute of time-triggered systems: "*If* n *nodes are already integrated, the integration of* n + 1 *node must not disturb the system*" [4].

From the communication service point of view, the cycle length deviation (600 ppm) of the stimuli stayed within the range tolerated by the FlexRay protocol and hence did not drive the clock synchronization out of range (like in 4), therefore they do not affect the application related network traffic. In practice this means that this test of the error detection mechanisms can be performed in parallel for each node of the system while the system operation is ongoing. The only requirement is a number of time slots for the tester to send its stimuli. Since existing mechanisms are exploited, no changes in the nodes under test are required. Like above the requirements on the tester are to be able to precisely control the position of the message to be sent and to precisely resolve the position of the received messages.

6. Conclusion

Given the economical and practical constraints that must be considered for an online test of an embedded automotive communication system we have elaborated and compared different test approaches that do not interfere with the ongoing application and that can be performed by a remote tester whose only access to the system under test is via the communication network. We have shown that even straightforward monitoring of the network traffic can yield an appreciable amount of diagnostic information, if it is supported by a careful interpretation that incorporates detailed protocol know-how. This is especially true for the transmit paths of the nodes within the system under test, since their outputs are directly observable. In contrast, diagnosis of the receive path is much more difficult, since neither its output nor its error status can be directly observed on the communication bus.

We have identified several possibilities to loop back this desired information to the communication network. The most attractive of these was the exploitation of the relation between reception status of sync messages and position of a subsequently transmitted message. This loop-back is very elegant and generic since it reuses existing standard mechanisms. Therefore, no additional service (and resource!) is required and this method can be applied with every FlexRay compliant system.

We have indicated and analyzed several approaches to make use of this loop-back: First the tester can add a few stimulus messages with the purpose to slightly manipulate the global time base. While this manipulation

remains completely transparent to the application, it allows the tester to distinguish, whether its stimulus messages have been properly received by the node under consideration. We have seen that no more than three (out of 2047 available) slots are required to implement this strategy. Second, the tester may take control of the clock synchronization process and apply a non conventional but still correct progression over time. While this approach may spoil the complete communication, it can also – if applied with appropriate care such that the associated distortion of the time scale is within tolerable bounds – enable a check of the error detection mechanisms within a node's receive path.

Future work will be directed towards an implementation and experimental assessment of the proposed approach. In particular the creation of a model for the "normal habits" of the clock synchronization behavior will require specific efforts. Furthermore a refinement of the fault injection concept will be elaborated.

Acknowledgements: The authors would like to acknowledge the contributions of the STEACS team in the preparation of concepts and tools for the experiments shown, and in particular that of Martin Horauer, who also contributed to an early version of this paper.

References

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

[2] G. Leen and D. Hefferman, "In-Vehicle Networks, Expanding Automotive Electronic Systems", in *IEEE Transaction on Computers*, January 2002, pp. 88–93.

[3] 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", in *Society of Automotive Engineers (SAE) 2001 World Congres*, March 2001.

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

[5] J. Tsai, Y.-D. Bi, S. Yang, and R. Smith, *Distributed Real Time System: Monitoring, Visualization, Debugging and Analysis*, Wiley-Interscience, New York, NY, USA, 1996.

[6] A. Benso and P. Prinetto, *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*, Kluwer Academic Publishers, Norwell, MA, USA, 2003.

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

[8] H. Kopetz, "Sparse Time versus Dense Time in Distributed Real-Time Systems", in *Proceedings of the 12th International Conference on Distributed Computing Systems*, June 1992, pp. 460–467.

[9] W. Schütz, "On the Testability of Distributed Real-Time Systems", in *Proceedings of the Tenth Symposium on Reliable Distributed Systems*, Oct. 1991, pp. 52–61.

[10] W. Elmenreich, G. Bauer, and H. Kopetz, "The Time-Triggered Paradigm", in *Proceedings of the Workshop on Time-Triggered and Real-Time Communication*, Dec. 2003, Manno, Switzerland.

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

[12] P. H. Bardell, W. H. McAnney, and J. Savir, *Built-in test for VLSI: pseudorandom techniques*, Wiley-Interscience, New York, NY, USA, 1987.

[13] R. Husemann and C. Pereira, "BR-Tool: a Real-Time Bus Monitoring and Validation System for Fieldbus-Based Industrial Automation Applications", in *IEEE Conference on Emerging Technologies and Factory Automation*, September 2003, pp. 145–152.

[14] C. Scherrer and A. Steininger, "Dealing with Dormant Faults in an Embedded Fault-Tolerant Computer System", in *IEEE Transaction on Reliability*, December 2003, pp. 512–522.

[15] H. Thane and H. Hansson, "Towards Systematic Testing of Distributed Real-Time Systems", in *Proceedings of the 20th IEEE Real-Time Systems Symposium*, December 1999, pp. 360–369.

[16] W. Schütz, "Fundamental Issues in Testing Distributed Real-Time Systems", *Real Time System Journal*, vol. 7, no. 2, pp. 129–157, 1994.

[17] 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.

[18] A. Kirschbaum, J. Becker, and M. Glesner, "Run-Time Monitoring of Communication Activities in a Rapid Prototyping Environment", in *Proceedings of the Ninth International Workshop on Rapid System Prototyping*, June 1998, pp. 52–57.

[19] M. El Shobaki and L. Lindh, "A Hardware and Software Monitor for High-Level System-on-Chip Verification", in *Proceedings of the International Symposium on Quality Electronic Design*, March 2001, pp. 56–61.

[20] D. Mahrenholz, "Minimal Invasive Monitoring", in *Proceedings of the Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, May 2001, pp. 251–258.

[21] D. Mahrenholz, O. Spinczyk, and W. Schroder-Preikschat, "Program Instrumentation for Debugging and Monitoring with AspectC++", in *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, April 2002, pp. 249–256.

[22] H. Heinecke and al., "AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E Architectures", in *Proceedings of Convergence, SAE-2004-21-0042*, March 2004.

[23] "FlexRay Communications Systems – Protocol Specification Version 2.1", FlexRay Consortium, 2005.

[24] E. Armengaud, A. Steininger, and M. Horauer, "Efficient Stimulus Generation for Remote Testing of Distributed Systems – The FlexRay Example", in L. L. Bello and T. Sauter, editors, *10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2005)*, volume 1, September 2005, pp. 763–770, Catania, Italy.