Optimizations of an Application-Level Protocol for Enhanced Dependability in FlexRay

Wenchao Li*, Marco Di Natale†, Wei Zheng*,
Paolo Giusto‡, Alberto Sangiovanni-Vincentelli*, Sanjit A. Seshia*

*EECS Department, UC Berkeley, CA, USA
wenchaol,zhengwei,alberto,sseshia@eecs.berkeley.edu

†Scuola Superiore S. Anna, Pisa, Italy
marco@sssup.it

‡General Motors, Palo Alto, CA, USA
paolo.giusto@gm.com

Abstract

FlexRay [9] is an automotive standard for high-speed and reliable communication that is being widely deployed for next generation cars. The protocol has powerful error-detection mechanisms, but its error-management scheme forces a corrupted frame to be dropped without any notification to the transmitter. In this paper, we analyze the feasibility of and propose an optimization approach for an application-level acknowledgement and retransmission scheme for which transmission time is allocated on top of an existing schedule. We formulate the problem as a Mixed Integer Linear Program. The optimization is comprised of two stages. The first stage optimizes a fault tolerance metric; the second improves scheduling by minimizing the latencies of the acknowledgement and retransmission messages. We demonstrate the effectiveness of our approach on a case study based on an experimental vehicle designed at General Motors.

I. Introduction

In future automotive applications, safety-critical applications such as the upcoming *X-by-Wire* features will need support from the underlying communication infrastructure. The increasing demand for bandwidth and the need for tight time predictability pose serious challenges to the current communication infrastructure such as the Controller Area Network (CAN) [2].

For safety critical sub-systems, the automotive industry currently refers to the IEC61508 standard [1] and is pursuing the definition of a tailored standard (ISO 26262). These standards provide a structured approach to assuring robustness against systematic and random faults. IEC61508 defines rules to achieve functional safety and identifies four levels of integrity (robustness) called Safety Integrity Levels (SILs) 1 to 4. For each SIL, the standard constrains the probability of a system-level failure. For SIL2 and SIL3 systems, the probability of failure per hour is, respectively, between 10^{-6} and 10^{-7} and between 10^{-8} and 10^{-9}. Current automotive systems are fail-safe: when a failure occurs, the system must move to a safe state [6]. Future X-by-wire systems will require to be fail-operational. Safety, communication bandwidth and determinism requirements motivate the need for a new communication infrastructure.

FlexRay [9] is a hybrid time-triggered communication protocol for automotive systems based on a fiber or copper medium. This protocol is a standard being developed by a consortium of major automotive manufacturers and Tier1 (automotive electronics) suppliers. It supports up to 10 Mb/s communication speed. The bus is assigned according to a time-triggered pattern: time is divided into communication cycles. Each cycle contains up to four segments: static, dynamic, symbol, and network idle time (*nit*), as shown in Figure 1. Clock synchronization is embedded into the standard and realized using part of the *nit* segment. Hence, it does not result in additional cost. Of the communication segments, the static part allows transmission of time-critical messages according to a periodic cycle in which the system allocates time slots to nodes for transmission of their outgoing messages. The number of static slots is fixed at design time and all slots have the same length. A static slot is assigned to a node for all cycles or is not assigned at all. Application data transmitted by a node may be less than the available data content of a slot. This under-utilization offers an opportunity for the implementation of redundancy for fault tolerance. Transmission time on the dynamic segment is assigned according to a mini-slot architecture. In the dynamic segment, each message has an associated frame ID. Dynamic slots are numbered with increasing IDs. A message can only be sent if its frame ID matches the current dynamic slot count. If no message is sent, the dynamic slot will collapse to one single mini-slot; otherwise, the dynamic slot will include the number of mini-slots that are needed to transmit the message. The combination of static and dynamic segments allows FlexRay to provide both time-triggered and event-triggered communication.

FlexRay also includes an optional dual-channel bus specification for increased reliability. Including bus guardians at the node- and star-level in the upcoming specification will in turn offer increased reliability and protection against timing faults. However, the full-scale adoption of FlexRay for safety critical applications is subject to the fulfillment of a number of additional implementation requirements at all levels of the communication stack, including the application layer, which is expected

978-3-9810801-5-5/DATE09 © 2009 EDAA

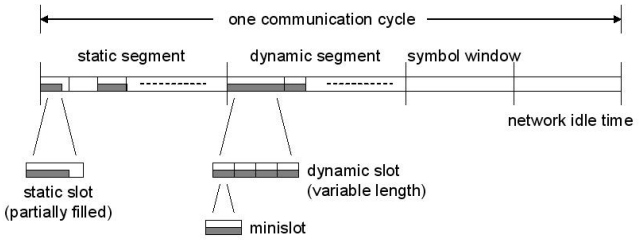

Fig. 1: A FlexRay Communication Cycle

to provide support for preventing message loss when necessary.

FlexRay has powerful error-detection mechanisms including clique detection and membership agreement. However, the foreseen error-management scheme instructs the receiver to discard a corrupted frame. Since the standard does not provide support for an acknowledgement mechanism (which exists in CAN), if an application requires a reliable communication, an acknowledgement and retransmission scheme must be implemented at the application level. The main challenge of implementing such a scheme is finding available transmission time in slots that can be used for acknowledgement and retransmission.

In this paper, we present an optimization-driven approach to the implementation of an acknowledgement and retransmission scheme on top of an existing FlexRay schedule with only static slot allocations. This means both slot size and slot ownership are predetermined. This choice, while allowing a simpler formulation of the problem, is also representative of a realistic design scenario, in which additional support for reliability must be obtained without changing the existing slot assignments to minimize changes to legacy sub-systems. In this context, a schedule that is a realistic representative of a safety-critical application is used as the baseline for the implementation and the optimization of the acknowledgement and retransmission scheme.

The paper is organized as follows: Section II provides the preliminary concepts. Section III summarizes our tool flow; Section IV presents a mathematical formulation of the problem and illustrates how the two-stage optimization is achieved; Section V presents the results on an experimental case study derived from a prototype vehicle designed by General Motors; Section VI provides a conclusion of this work and ideas for future extensions.

II. Preliminaries

There has been extensive work on schedule optimization for communication protocols such as CAN [11]. More recently, Zheng et al. [10] presented an optimization framework, based on an MILP formulation, that selects task and message activation models in a real-time CAN-based distributed automotive system. Davare et al. [3] used a mixed-integer geometric programming optimization procedure to assign periods for tasks and messages in similar CAN-based systems.

In comparison, the work done on schedule optimization for FlexRay is relatively scarce. Ding et al. [4] proposed a Genetic Algorithm for statically scheduling FlexRay systems. More notably, Pop et al. [7] have presented timing analysis techniques for messages transmitted in both the static and the dynamic segments of a FlexRay cycle. While their work focuses on schedulability analysis, little attention has been paid to fault tolerance. To the best of our knowledge, the work described in this paper is the first one that formally evaluates scheduling techniques for application-level fault tolerance in FlexRay - the proposed algorithm is a novel attempt at optimizing fault tolerance on top of an existing schedule.

We consider a distributed architecture consisting of a set of n ECU nodes connected by one FlexRay bus. Redundancy in communication can be provided by having an extra bus, but at the cost of additional hardware. The basic node architecture consists of a host processor, a FlexRay communication controller, an optional bus guardian and a bus driver. In this work, the details of the implementation of each component are abstracted by a set of tasks and signals, as described in the following.

Computations on each host processor are represented by tasks. Each task has an offset, a period, and a worst-case execution time (WCET). In our formulation, we treat different activations of the same task at different time instances as different jobs with different activation offsets. A signal is defined as an information flow between two tasks (which may be executed on different ECUs). The direction of the flow indicates a communication and precedence dependency. Multiple signals from the same ECU can be packaged into one message. Therefore, a message is the basic unit of communication on the bus (note that a message must have a single source ECU but may have multiple destination ECUs). This structure is illustrated in Figure 2 (arrows represent information flows, i.e. signals). Each message is allocated into a slot that is owned by its source ECU in a communication cycle. Note that ownership of a slot, once assigned to an ECU, extends to all cycles.

The *application cycle* or *hyperperiod* H is defined as the least common multiple of the periods of all tasks. Inside the hyperperiod, each job (task instance) is considered as an atomic scheduling entity. The scheduling problem consists of planning the execution of jobs and the transmission of signals into the available slots inside H. The scheduling of the FlexRay communication consists of mapping the tasks and signals defined in the application cycle onto a set of communication cycle instances (up to 64, according to the standard. Figure 3 shows a case with 4 communication cycles). Each communication cycle (a power-of-two submultiple of the application cycle) has its own scheduling table, that assigns signals to slots. Additional tables may optionally define the scheduling of tasks by time-triggered RTOS on the ECUs.

In this work, we consider only the static part of an existing FlexRay schedule. The basic parameters of a FlexRay architecture such as the static window size and the slot size are assumed to have been determined *a priori*.

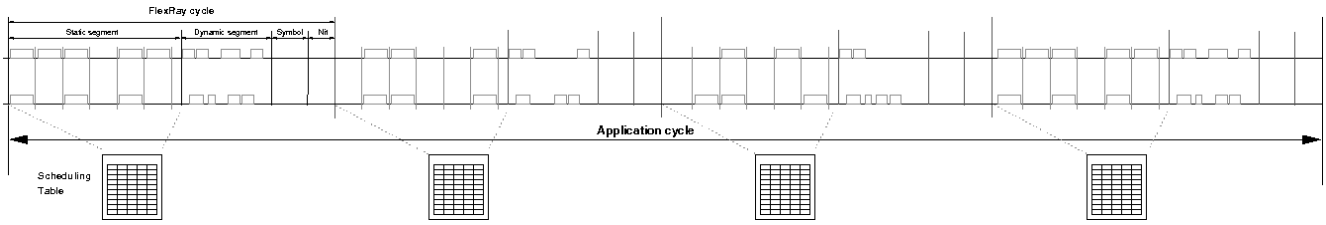

Fig. 3: Application cycle and FlexRay cycle

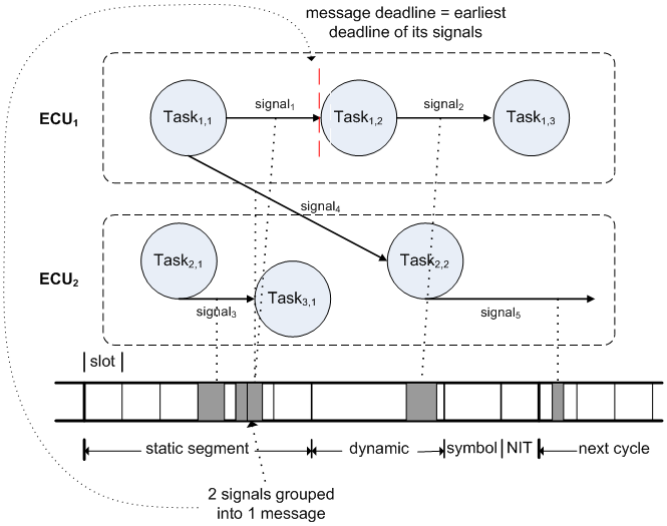

Fig. 2: Task Dependency Graph

Given the offsets and WCETs of all the receiving tasks of each message, a preprocessing stage computes the deadline constraints of each task and message. The deadline is computed as the earliest start time of the successors and it is approximated as an integer number of communication slots.

Given a particular architecture instance of a FlexRay bus and a task dependency graph such as the one shown in Figure 2, the objective of a FlexRay scheduler is to find a feasible (if possible) schedule for message allocations by suitably assigning slots to ECUs. In addition, the scheduler may try to optimize with respect to computation and communication latencies. We model the occurrence of a fault at the data-link level. Under an acknowledgement and retransmission scheme, if a message is detected as faulty by some receiving node(s), the receiving node(s) will send a *NACK* message (a fault has occurred) to the source ECU. If there is at least one NACK message, the source ECU will retransmit the message. In addition, we assume that messages have timing constraints and the retransmitted message has the same deadline as the original message. Faults sometimes may behave inconsistently on the content of a message for different nodes. In the worst case, a message can be faulty for all its receiving nodes but not its sending node. When a fault occurs, our scheme will force all receiving nodes to discard the faulty message and send negative acknowledgements to the sending node, which will in turn trigger a retransmission. Furthermore, we assume that faults can occur in any message within the cycle. An acknowledgement is abstracted into a 1-bit message for each destination ECU. In addition, acknowledgements (as well as retransmitted messages) can be *piggy backed* in an already occupied slot as long as they can be accommodated while observing slot ownership constraints.

In order to justify the previous assumptions and motivations for this work, fault data (probability measures associated with each fault type) for the FlexRay bus are needed. Unfortunately, the standard is completing its definition and the first applications have just been put into the market in 2008. Therefore, given the novelty of the standard and the scarcity of actual implementations, experimental data are not available. One possibility is to use the fault rate data available for CAN to understand what are the challenges for high-speed FlexRay communication. In doing this evaluation, however, it is necessary to remember that the FlexRay physical layer has been defined to achieve a reliability significantly higher than the CAN bus.

In [5], bit error rates (BER) have been determined for CAN communication for three different environment types. The values are, for a benign environment, 3×10^{-11}, for a normal environment 3.1×10^{-9} and, for an aggressive environment 2.6×10^{-7}. For the example presented in the experimental part of this study, considering that the CAN CRC protects also the header and that we need to consider errors in the 24 CRC bits as well as in the remaining frame bits, there are 14075 bits transmitted at each cycle that could possibly lead to an error in a CAN implementation. The probability of having more than one bit error per cycle is correspondingly 8.33×10^{-15} for a benign environment, 2.64×10^{-11} for a normal environment and 1.85×10^{-7} for an aggressive environment. However, the probability of having no errors in a cycle (1 ms in our example) is from 7.02×10^{-8} (benign) to 6.08×10^{-4} (aggressive). If one error per cycle is masked by an application-layer mechanisms like the one presented here, the resulting number of errors per hour is from 3×10^{-8} to 4.86×10^{-1}, which may be acceptable considering that FlexRay is constructed to be significntly more reliable. However, if no mechanism is in place for protecting the application against single errors, the number of errors per hour becomes significantly larger and unacceptable, between 0.22 and 1 error per hour, depending on the quality of the environment.

Our problem therefore is the following. Given an exist-

ing schedule, we try to optimize ownership assignments to unassigned slots so that fault tolerance can be maximized. We formulate the optimization problem as a *Mixed Integer Linear Program* (MILP). Details of the formulation are given in Section IV.

III. Tool Flow

The proposed flow of development for a robust FlexRay configuration is the following. Given a task dependency graph, and a predetermined FlexRay architecture instance, first the designer determines a feasible schedule. This portion of the flow is outside the scope of this work and assumed to have already been performed. Given the existing schedule, we then proceed to incrementally build on it to provide additional reliability. A pre-processing stage automatically extracts deadline constraints on the messages from the task dependency graph. The deadline of any message is the earliest deadline of its constituent signals. A signal must arrive before its receiving task is scheduled to start.

The optimization stage is decomposed into two sub-stages, where the first sub-stage optimizes the *fault recovery rate* (defined in Section IV-A) and the second sub-stage optimizes the execution of the acknowledgement and retransmission scheme. At the end, we evaluate the performance tradeoffs with this additional reliability support.

IV. Mathematical Formulation

A. Metric, Parameters and Variables

We use the following fault tolerance metric.

The fault recovery rate is the percentage of faulty messages guaranteed to be retransmitted before their deadlines.

The existing system and the FlexRay scheduling configurations are expressed by the following parameters and variables.

Parameters:

ECUs. The set of ECUs is labeled as $E = \{ECU_i\}$.

Messages are labeled as M_i. The number of messages is n_m. For each message M_i we associate

- a size, in bits w_i.
- an allocation at slot ms_i and cycle mc_i.
- a deadline constraint (expressed in slot units) d_i.
- a source ECU se_i.
- a set of destination ECUs de_i.

FlexRay Cycles are denoted as Cy_i. The number of cycles in an application cycle is n_c. Each cycle has n_s static slots.

Static Slots are labeled as S_i. For each slot S_i, we indicate as $Owned_i$ the index of the ECU (if defined by the predetermined schedule) that has the transmission rights of slot i (a slot can be initially not assigned). The slot size l_s is the maximum number of bits that a slot can accomodate (the same for all the static slots).

A **Schedule** matrix of dimension $n_s \times n_c$ is given, where each entry sch_{ij} is the size of the message that occupies the corresponding slot, 0 if the slot is not used by any message.

Variables:

For each message M_i we define

- a binary variable f_i indicating whether a retransmission is scheduled if faulty.
- a pair rs_i, rc_i of integers, if a retransmission is performed at slot rs_i and cycle rc_i.
- for each possible destination ECU j of M_i, a pair of integers as_{ij}, ac_{ij}, indicating the slot and cycle index that are optionally used for transmitting the acknowledgement sent by ECU_j for M_i.

For each slot S_i, a binary variable own_{ij} indicates whether S_i is assigned to ECU_j. If a slot is initially unassigned, then $own_{ij} = 0$ for all j.

A set of binary variables ack_{ijkl} is defined, with value 1 iff sink j of faulty message i place its acknowledgement in slot k cycle l.

A binary variable ret_{ikl} is defined for each (message, slot, cycle) triple, with value 1 iff retransmission of faulty message i is placed in slot k cycle l.

B. Constraints

We describe here the constraints in the MILP. For each constraint, we first give a mathematical description of the constraint, using standard logical connectives. Then, we state the linear constraint that encodes this mathematical description.

Acknowledgments are placed iff the original message is protected against faults $\forall i, j : \{1 \leq i \leq n_m, j \in de_i\}$ and M is a large enough constant,

$$f_i \leq as_{ij} \leq M \times f_i \quad (1)$$
$$f_i \leq ac_{ij} \leq M \times f_i \quad (2)$$

Retransmission is placed iff the original message is protected against faults $\forall i : \{1 \leq i \leq n_m\}$,

$$f_i \leq rs_i \leq M \times f_i \quad (3)$$
$$f_i \leq rc_i \leq M \times f_i \quad (4)$$

Acknowledgements must follow transmissions For every faulty message that is to be retransmitted, the acknowledgement slot must occur after the slot to which the message is allocated.

Hence, $\forall i \, s.t. \, 1 \leq i \leq n_m, \forall j \in de_i, \big(f_i \rightarrow (ms_i + (mc_i - 1)n_s \leq as_{ij} + (ac_{ij} - 1)n_s)\big)$

The constraint above is represented as a linear inequality with the standard method that makes use of a large constant M to model disjunctive constraints:

$$ms_i + (mc_i - 1)n_s - as_{ij} - (ac_{ij} - 1)n_s \leq M(1 - f_i) \quad (5)$$

Retransmissions must follow acknowledgements For every faulty message that is to be retransmitted, the retransmission slot must occur after the corresponding acknowledgement slot.
Hence, $\forall i\, s.t.\, 1 \leq i \leq n_m, \forall j \in de_i,\ \big(f_i \rightarrow (as_i + (ac_i - 1)n_s \leq rs_{ij} + (rc_{ij} - 1)n_s)\big)$

The corresponding linear inequality is:

$$as_{ij} + (ac_{ij} - 1)n_s - r_i - (r_i - 1)n_s \leq M(1 - f_i) \quad (6)$$

Slot Utilization Acknowledgements and retransmission must be placed in a slot with enough space to accommodate them. $\forall i, k, l : \{1 \leq i \leq n_m, 1 \leq k \leq n_s, 1 \leq l \leq n_c\}$,

$$\big(\sum_{j \in de_i} ack_{ijkl}\big) + ret_{ikl} \times w_i + sch_{kl} \leq l_s \quad (7)$$

Already assigned slots are set to their corresponding owner ECUs $\forall i\, s.t.\, Owned_i \neq 0$ (0 means unassigned),

$$own_{i,Owned_i} = 1 \quad (8)$$

A slot can only be owned by one ECU The same slot for all cycles can only be owned by one ECU. $\forall i : \{1 \leq k \leq n_s\}$,

$$\sum_{j \in ecu} own_{kj} = 1 \quad (9)$$

Acknowledgments must be sent according to slot ownership Acknowledgements must be sent in slots that are owned by the corresponding ECUs. $\forall i, j, k, l : \{1 \leq i \leq n_m, j \in de_i, 1 \leq k \leq n_s, 1 \leq l \leq n_c\}$,

$$ack_{ijkl} \leq own_{kj} \quad (10)$$

Retransmissions must be sent according to slot ownership Retransmissions must be sent in slots that are owned by the corresponding ECUs. $\forall i, j : \{1 \leq i \leq n_m, j \in se_i, 1 \leq k \leq n_s, 1 \leq l \leq n_c\}$,

$$ret_{ikl} \leq own_{kj} \quad (11)$$

Deadline constraints Retransmissions must be sent before the deadlines (constraints on acknowledgements are implicit given the precedence constraints between acknowledgements and retransmission.) $\forall i : \{1 \leq i \leq n_m\}$,

$$rs_i + (rc_i - 1) \times n_s \leq d_i \quad (12)$$

Intermediate Variables Intermediate Variables for acknowledgements and retransmissions are constrained correspondingly.

Acknowledgements $\forall i, j : \{1 \leq i \leq m, j \in de_i\}$.

$$\sum_{1 \leq k \leq n_s, 1 \leq l \leq nc} ack_{ijkl} = f_i \quad (13)$$

$$\sum_{1 \leq k \leq n_s, 1 \leq l \leq nc} ack_{ijkl}(k + l \times n_s) = as_{ij} + ac_{ij} \times n_s \quad (14)$$

Retransmissions $\forall i : \{1 \leq i \leq m\}$.

$$\sum_{1 \leq k \leq n_s, 1 \leq l \leq nc} ret_{ikl} = f_i \quad (15)$$

$$\sum_{1 \leq k \leq n_s, 1 \leq l \leq nc} ret_{ikl}(k + l \times n_s) = rs_i + rc_i \times n_s \quad (16)$$

C. Two-stage Optimization

We first optimize the number of fault recovery opportunities.

$$maximize : \sum_{1 \leq i \leq n_m} f_i \quad (17)$$

At the second stage, given the set of messages M that can be recovered, we optimize in an enumerative way the placement of acknowledgements and retransmission such that the retransmission happens as soon as possible for each of these messages.

$$\forall_{i \in M}, minimize :\ rs_i + (rc_i - 1) \times n_s. \quad (18)$$

V. Case Study

The FlexRay schedule in Figure 4 is obtained from an *X-by-Wire* application from General Motors. The number on each occupied slot corresponds to the weight of the corresponding message. The application has 10 ECUs interconnected by one single FlexRay bus. There are a total of 56 tasks and 78 messages in a span of 8 cycles (this is the application cycle of the schedule). The scheduler gives a total of 22 slots, each with a size of 192 bits. Out of these 22 static slots, 18 slots have already been assigned by the scheduler (in bold). For example, in Figure 4, $slot_1$ is assigned to ECU_1.

The mathematical formulation is encoded in AMPL [8]. The ILOG CPLEX 11.0.0 optimization solver is used on an i686 server running Linux. The average runtime for the first stage optimization is $13.1s$. We first automatically extract the deadline constraints from the raw data. The resulting deadline constraints on each message are in terms of slots (starting from the beginning of the application cycle). With these additional deadlines, we then generate a data file for the AMPL program.

		1	2	3	4	5	6	7	8
1	**ECU1**			32				160	
2	**ECU2**			32				160	
3	**ECU3**			32				160	
4	**ECU4**			32				160	
5	**ECU5**	160	160	160	160	160	160	160	160
6	**ECU6**	160	160	160	160	160	160	160	160
7	**ECU7**	192	192	192	192	192	192	192	192
8	**ECU8**	192	192	192	192	192	192	192	192
9	**ECU9**					192			
10	**ECU9**					192			
11	*ECU8*								
12	*ECU5*								
13	**ECU7**	1	1	1	1	1	1	1	1
14	**ECU8**	64	64	64	64	64	64	64	64
15	**ECU9**				120				
16	**ECU5**	112	112	112	112	112	112	112	112
17	**ECU6**	112	112	112	112	112	112	112	112
18	*ECU8*								
19	**ECU9**								128
20	**ECU10**								128
21	**ECU9**		1						
22	*ECU5*								

Fig. 4: Schedule with Weights (8 cycles and 22 slots)

The first stage optimization results in a maximum of 43 messages being resilient to single fault using the acknowledgement and retransmission scheme. The optimal

slot assignment assigns $slot_{11}$ to ECU_8, $slot_{12}$ to ECU_5, $slot_{18}$ to ECU_8 and $slot_{22}$ to ECU_5 (the ones in italic). We compared the optimal solution with the alternatives of not using the unassigned slots (worst case) or randomly assigning them. Using our *MILP* formulation, 55.1% of possibly faulty messages can be recovered. However, if we randomly assign the unassigned slots, only 40.8% are recoverable. If we do not use the unassigned slots at all, only 33.3% are recoverable. Hence, effective use of unassigned slots can significantly increase fault tolerance.

Figure 5 shows how the fault recovery rate changes as we increase the load. Messages are randomly added to both occupied and unoccupied slots. Fault recovery rate increases initially because the original schedule still has enough vacant transmission time for the additional acknowledgements and retransmissions. However, the rate drops as the number of added messages increases to 15. This is because some of the unassigned slots that we originally use for acknowledgements and retransmissions have become unavailable as they are now assigned to the source ECUs of the added messages. Intuitively, the more unassigned slots and vacant transmission time we have, the more slack we have for acknowledgements and retransmissions.

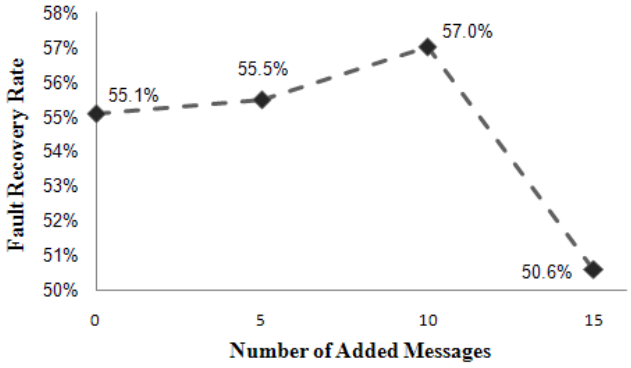

Fig. 5: Fault Recovery Rate vs. Load

At the second stage, we fix the slot assignments using the optimal solution found in the first stage. For each faulty message that can be recovered, we optimize the placement of its acknowledgements and retransmission such that they are sent as soon as possible by using the second objective function in Section IV-C. For example, in Figure 4, if $message_1$ (in $slot_5$ $cycle_1$) becomes faulty, the scheme will place an acknowledgement from ECU_7 in $slot_{13}$ and an acknowledgement from ECU_8 in $slot_{14}$. ECU_5 then retransmits the message in $slot_{22}$.

VI. Conclusion and Future Work

We have presented a MILP formulation for implementing an application-level acknowledgement and retransmission scheme in FlexRay communication. The formulation allows the scheme to be optimized for fault tolerance. The choice of a fixed schedule consisting of only static slots simplifies the formulation of the problem. However, this represents a realistic setting in which the additional support for reliability must be obtained with minimal changes to the existing subsystems. In addition, post-assigning slots through an optimization procedure with an application-level acknowledgement and retransmission scheme is consistent with the FlexRay protocol. In terms of architecture parametrization, we have limited insight on the tradeoffs with fault tolerance. Even so, we hypothesize that a small idle dynamic window may be useful for increasing fault tolerance because it can guarantee that acknowledgements and retransmission are placed at the end of a communication cycle in the worst case. Formulating the problem as an optimization problem offers advantages such as ease of use and the guarantee of optimal solutions with respect to some objective functions. However, this procedure can be computationally expensive and thus may not scale to larger designs. We are currently working on investigating how much more fault tolerance we can harness if the optimization is done at schedule time. We are also working on a variant of this formulation in which messages are recovered based on some priority assignments. In the future, we would like to explore other fault tolerant mechanisms such as forward error correction.

Acknowledgements. The authors gratefully acknowledge the support of the Hellman Family Faculty Fund and the support of the Gigascale Systems Research Focus Center, one of five research centers funded under the Focus Center Research Program. The authors would also like to acknowledge the support of ArtistDesign network of Excellence and the STREP project COMBEST.

References

[1] Cei international standard. iec 61508.
[2] R. Bosch. Can specification, v. 2.0., 1991.
[3] A. Davare, Q. Zhu, M. D. Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. Period optimization for hard real-time distributed automotive systems. In *DAC '07: Proceedings of the 44th annual conference on Design automation*, pages 278–283, New York, NY, USA, 2007. ACM.
[4] S. Ding, N. Murakami, H. Tomiyama, and H. Takada. A ga-based scheduling method for flexray systems. In *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*, pages 110–113, New York, NY, USA, 2005. ACM.
[5] J. Ferreira, A. Oliveira, P. Fonseca, and J. A. Fonseca. An experiment to assess bit error rate in can. In *3rd Intl Workshop on Real-Time Networks*, Catania, Italy, June 2004.
[6] T. Forest, A. Ferrari, G. Audisio, M. Sabatini, A. Sangiovanni-Vincentelli, and M. D. Natale. Physical architectures of automotive systems. In *DATE '08: Proceedings of the conference on Design, automation and test in Europe*, pages 391–395, New York, NY, USA, 2008. ACM.
[7] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing analysis of the flexray communication protocol. *Real-Time Syst.*, 39(1-3):205–235, 2008.
[8] www.ampl.com. A mathematical programming language.
[9] www.flexray.com. Flexray specification.
[10] W. Zheng, M. D. Natale, C. Pinello, P. Giusto, and A. Sangiovanni-Vincentelli. Synthesis of task and message activation models in real-time distributed automotive systems. In *DATE'07: Proceedings of the Design, Automation and Test in Europe Conference*, Nice, France, April 2007.
[11] K. M. Zuberi and K. G. Shin. Scheduling messages on controller area network for real-time cim applications. *Robotics and Automation, IEEE Transactions on*, 13(2):310–316, April 1997.