Performance analysis of FlexRay-based systems using Real-Time Calculus, Revisited

Devesh B. Chokshi
Computer Engineering and Networks Laboratory
ETH Zurich
8092 Zurich, Switzerland
devesh.chokshi@tik.ee.ethz.ch

Purandar Bhaduri
Dept. of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati 781039, India
pbhaduri@iitg.ernet.in

ABSTRACT

The FlexRay protocol [4] is likely to be the *de facto* standard for automotive communication systems. Hence, there is a need to provide hard performance guarantees on properties like worst case response times of messages, their buffer requirements, end-to-end latency (for example, from sensor to actuator), etc., for FlexRay based systems. The paper [11] provides an analysis for finding worst case response times of the messages transmitted on the FlexRay bus, but the analysis is done using ILP formulation and is thus computationally expensive. The paper [5] models the FlexRay in the analytic framework of Real-Time Calculus [12, 3] and is compositional as well as scalable. In this paper, we show that the analysis of [5] may lead to results that are over optimistic; in particular, we show that obtaining the "upper service curves" is not trivial and does not follow the reasoning of the "lower service curves" which the authors obtain. We also provide tighter "lower service curves" than that of [5]. Finally we show that our model allows the messages to be of variable size which is not the case with [5].

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems; C.4 [**Performance of systems**]: Design studies and modeling techniques

General Terms

Performance, Design

Keywords

FlexRay, Real-Time Calculus

1. INTRODUCTION

The constraints imposed by application requirements in the automotive domain lead to complex distributed architectures consisting of multiple electronic control units (ECUs) communicating via a bus. Applications are partitioned into tasks mapped onto different ECUs with message passing between these tasks. Modern high-end cars may contain around 70 ECUs with up to 2500 messages for communication between them [5], which leads to bus-based system. This has led to the development of several communication protocols like CAN [2], LIN [10], etc.

Communication protocols are broadly classified as time-triggered (message triggered at specific moments) or event triggered (message triggered by some event). The main disadvantage of time-triggered protocols like TTP [8] is the lack of flexibility and lower bandwidth utilization, especially for aperiodic messages, but they have the advantage of predictability. On the other hand, event-triggered protocols are more flexible and have high bandwidth utilization. But the analysis of event-triggered protocols is difficult without making pessimistic assumptions as they are dynamic, due to which they pose a challenge for safety-critical systems which require hard bounds on properties like message response time, buffer requirements, end-to-end delay, etc.

Therefore, there is now growing interest in hybrid protocols such as TTCAN [9], Byteflight [7] and FlexRay, that support both time-triggered and event-triggered communications. FlexRay is promoted by a large number of automotive companies and is likely to be the *de facto* standard for in-vehicle communication. Hence there has been a lot of interest in performance analysis of FlexRay based systems.

Related work: The paper [11] formally models the FlexRay bus i.e., both ST and DYN segments. It propose the techniques of finding worst case response time of the event-triggered message mapped on to the DYN segment using the ILP formulation and this problem is shown to be similar to bin covering problem, and hence this technique is computationally expensive. The paper also provides certain pessimistic heuristics to address the issue of scalability.

Using the framework of Real-Time Calculus (RTC), [5] presents a compositional analysis of FlexRay based systems, and it can be used to determine various performance criteria such as worst-case response time of messages mapped onto the FlexRay bus, end-to-end delay from sensor to actuator, buffer requirements for the tasks and messages, etc.

Our contribution: In this paper, we show that the analysis of [5] may lead to overly optimistic results. Particularly, we show that obtaining the "upper service curves" is not trivial and does not follow the reasoning of the "lower service curves" which the authors of [5] obtain. We also provide tighter "lower service curves" than that of [5]. Moreover, [5] assumes that all the messages are of fixed sizes. But we

SAC'10 March 22-26, 2010, Sierre, Switzerland.

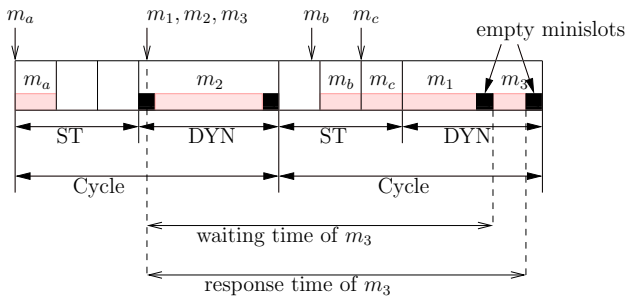


Figure 1: FlexRay cycles

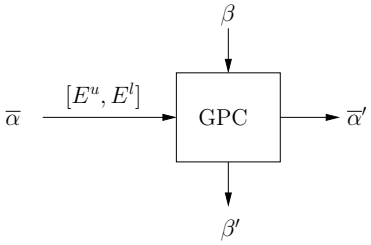


Figure 2: Greedy processing component

show that our model allows the messages to be of variable sizes specified in terms of best case and worst case sizes. Note that the analysis of the messages mapped onto the ST segment is known as it is like TDMA, so we discuss how to analyze DYN messages only in this paper.

2. THE FLEXRAY PROTOCOL

In this section, we show how the messages generated by ECUs are transmitted on the FlexRay bus. Note that throughout the paper, we will consider the system model and assumptions of [5] which we describe here in brief.

In the FlexRay protocol, each communication cycle is composed of a ST segment and a DYN segment and such a cycle is repeated periodically. The lengths of ST and DYN segments may not be equal but they are fixed across communication cycles during the design phase.

The ST segment consists of a fixed number of equal-length slots. Each such slot is allocated to a particular message and a message is allowed to be send only during its allocated slot. If the message is not ready at the beginning of its allocated slot, then this slot is empty and other messages are not allowed to be sent in this slot.

The DYN segment consists of equal-length "minislots". Messages mapped onto DYN segment are assigned fixed priorities. At the beginning of the DYN segment, the highest priority message is allowed to be send. The length of such message can be arbitrary long but it must fit within one DYN segment. However, if this message is not ready at the beginning of the slot, then one minislot goes empty. In either case, the access is then given to a next- highest priority message, which is sent if it is ready and if it fits into the remaining portion of DYN segment. Otherwise, one minislot goes empty. This process is repeated for all the messages or till the end of the DYN segment whichever is early.

As an example, consider the messages m_a, m_b and m_c mapped onto the ST segment and messages m_1, m_2 and m_3 (m_1 has highest priority) mapped onto the DYN segment. Figure 1 shows two consecutive FlexRay cycles. The arrival of the message is shown by a downward arrow. Considering ST messages, in the first cycle, only m_a is sent as it is ready and the slots corresponding to m_b and m_c go empty. In the second cycle, m_b and m_c are sent. Considering DYN messages, in the first cycle, m_1 arrives just after its turn, so it is not sent in the first cycle, and one minislot goes empty. Then, m_2 is sent. The message m_3 is not sent in the first cycle as it cannot be accommodated in the remaining DYN segment. In the second cycle, m_1 is sent. One minislot goes empty as m_2 has not arrived. Then, m_3 is sent.

3. REAL-TIME CALCULUS (RTC)

The key concepts in the framework of Real-Time Calculus [12, 3] are (i) the modeling of the arrival pattern of tasks (or the *event model*) which generates demands on the resources, (ii) the modeling of the service offered by the resources to the tasks (i.e., the *resource model*), and (iii) the component model (processing semantics). It can be used to derive hard upper and lower bounds of various performance criteria such as maximum end-to-end delay experienced by an event stream or buffer requirements.

Arrival Curves: The event model is captured by the notion of arrival curves. Let $R[s,t)$ be the number of events that arrive in the time interval $[s,t)$. Then R, the upper arrival curve $\overline{\alpha}^u$ and the lower arrival curve $\overline{\alpha}^l$ are related by $\overline{\alpha}^l(t-s) \leq R[s,t) \leq \overline{\alpha}^u(t-s), \forall s \leq t$ with $\overline{\alpha}^l(0) = \overline{\alpha}^u(0) = 0$. We write $\overline{\alpha} = [\overline{\alpha}^u, \overline{\alpha}^l]$ and call it the arrival curve.

Service curves: The resource model is captured by the notion of service curves. Let $S[s,t)$ be the number of events that a resource can service in the time interval $[s,t)$. Then S, the upper service curve $\overline{\beta}^u$ and the lower service curve $\overline{\beta}^l$ are related by $\overline{\beta}^l(t-s) \leq S[s,t) \leq \overline{\beta}^u(t-s), \forall s \leq t$ with $\overline{\beta}^l(0) = \overline{\beta}^u(0) = 0$. We write $\overline{\beta} = [\overline{\beta}^u, \overline{\beta}^l]$ and call it the service curve.

Arrival and service curves can also be described in terms of the amount of resources, such as the number of processing or communication cycles, instead of number of events as above. The *resource based service curve* $\beta(\Delta)$ denotes the resource units available in any time interval of length Δ. Similarly, the *resource-based arrival curve* $\alpha(\Delta)$ denotes the requests in terms of resource units that arrive in any time interval of length Δ.

Greedy Processing Component: In Real-Time Calculus, the greedy processing component (GPC) [3, 13, 6] is an abstract component that is triggered whenever an event is available on the input event stream (described by the arrival curve $\overline{\alpha}$) and produces a single output event stream (described by $\overline{\alpha}'$). At every event arrival, a task is instantiated to process the incoming event. Events are processed in a greedy fashion in first-in-first-out order, while being restricted by the availability of processing resources described by the service curve β.

Let E^u and E^l denote the execution demand in terms of the maximum and minimum resource units required for processing one event. Then the GPC can be modeled as:

$$\overline{\alpha}'^u = \lceil \min\{(\overline{\alpha}^u \otimes \overline{\beta}^u) \oslash \overline{\beta}^l, \overline{\beta}^u\} \rceil \quad (1)$$

$$\overline{\alpha}'^l = \lfloor \min\{(\overline{\alpha}^l \oslash \overline{\beta}^u) \otimes \overline{\beta}^l, \overline{\beta}^l\} \rfloor \quad (2)$$

$$\beta'^u = \max\{(\beta^u - \alpha^l)\overline{\oslash}0, 0\} \quad (3)$$

$$\beta'^l = (\beta^l - \alpha^u)\overline{\otimes}0 \quad (4)$$

where β' denotes the remaining service available to process other event streams , and the workload transformations are $\overline{\beta}^u = \beta^u/E^l$, $\overline{\beta}^l = \beta^l/E^u$, $\alpha^u = E^u.\overline{\alpha}^u$, and $\alpha^l = E^l.\overline{\alpha}^l$. See [1] for the definitions of $\otimes$, $\oslash$, $\overline{\otimes}$, and $\overline{\oslash}$. Refer to [3, 13] for a discussion of these results on the GPC.

The worst case response time WR (time between activation and completion) experienced by any event on the event stream and the maximum number of events in the input queue *Buffer* can be determined as follows [3, 13]:

$$WR \leq Del(\overline{\alpha}^u, \overline{\beta}^l) \quad (5)$$

$$Buffer \leq Buf(\overline{\alpha}^u, \overline{\beta}^l) \quad (6)$$

where $Del(\alpha^u, \beta^l) = \sup_{\Delta \geq 0}\{\inf\{\mu \geq 0 : \alpha^u(\Delta) \leq \beta^l(\Delta + \mu)\}\}$ and $Buf(\alpha^u, \beta^l) = \sup_{\lambda \geq 0}\{\alpha^u(\lambda) - \beta^l(\lambda)\}$.

The end-to-end response time r experienced by an event with upper arrival curve $\overline{\alpha}^u$ that is processed on N consecutive GPCs with lower service curves $\overline{\beta}^l_1, \ldots, \overline{\beta}^l_N$ is [13]:

$$r \leq Del(\overline{\alpha}^u, \overline{\beta}^l_1 \otimes \ldots \otimes \overline{\beta}^l_N) \quad (7)$$

4. FLEXRAY MODEL OF [5]

In this section, we briefly describe the model of FlexRay given by [5] and in section 4.1, we show that obtaining the upper service offered to a message is not trivial and does not follow the reasoning of the lower service given by [5]. In section 5, we revise the FlexRay model of [5].

Suppose there are n messages $m_1, \ldots, m_n$ mapped onto DYN segment (m_1 has the highest priority). For now, assume that message m_i requires k_i minislots. Let the length of the DYN segment be k minislots (or d time units) and the length of the communication cycle be p time units. Each minislot is ms time units long. Assume that $\beta = [\beta^u, \beta^l]$ is the bound on the service (expressed in terms of minislots in a given time interval) offered by the unloaded DYN segment (i.e., the total service to all DYN messages). We are interested in finding out $\beta_i = [\beta^u_i, \beta^l_i]$, which is the service actually offered to the message m_i.

According to [5], to obtain β^l_1, i.e. the lower service available to the highest priority message m_1 mapped onto DYN segment, β^l is transformed using the following steps:

1. Extract k_1 minislots during each communication cycle from β^l. Nullify the communication cycles containing less than k_1 minislots. This is to model that in any communication cycle, at most k_1 minislots are available to m_1 (as at most one instance of m_1 is allowed to be sent in one DYN segment). See Figure 3(a).

2. Discretize the service obtained from step 1. This is because a message can not straddle two communication cycles. See Figure 3(a).

3. Shift the service obtained after step 2 by d time units. This is to model that if a message is ready just after its turn, it has to wait for the next communication cycle. See Figure 3(b).

The other properties (like empty minislot) are considered later.

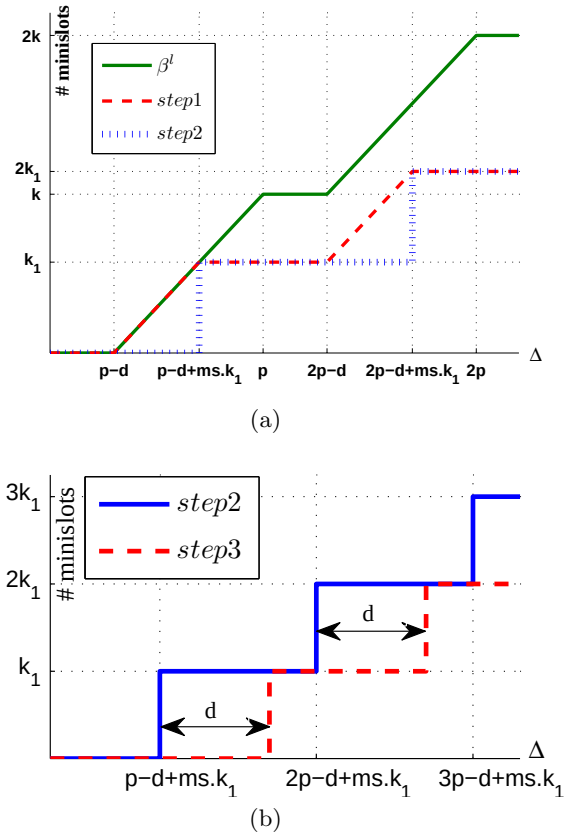


Figure 3: (a) Steps 1 and 2 and (b) Step 3 to obtain β^l_1 [5]

4.1 Discussion on the upper service available to a message

The paper [5] does not discuss on how to obtain the upper service available to a message. To obtain the upper service curve, clearly, step 3 as discussed above is not required. However, it is not obvious whether one can obtain upper service curve using the other two steps. We now show that the upper service curve obtained by using any of the following two methods will lead to incorrect results:

- **Method 1**: use steps 1 and 2 as discussed above
- **Method 2**: use step 1 only as discussed above

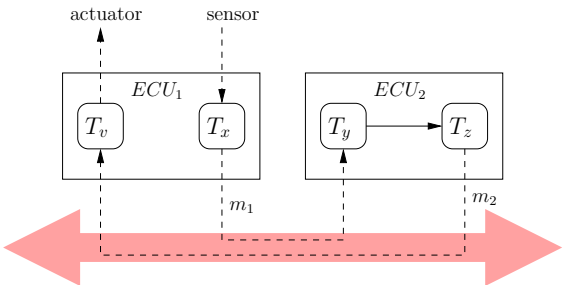


Figure 4: Example system

Consider a system having two ECUs (both use fixed priority preemptive scheduling) and a FlexRay bus as shown in Figure 4. Task T_x (highest priority) on ECU_1 generates message m_1 (highest priority) which is mapped onto the DYN segment of the FlexRay bus. Task T_y (highest priority) on ECU_2 is activated by the reception of m_1. Task T_z on ECU_2 is activated by the completion of T_y, which sends message m_2 through DYN segment to task T_v. Suppose the FlexRay cycle length is 10 time units and the DYN segment length is 7 time units. Assume that the length of one minislot is 1 time unit. Task T_x on ECU_1 is activated periodically with a period of 21 time units, and T_x requires an execution time of fixed 3 time units. Task T_y requires an execution time of fixed 2 units. Assume that the length of one processor cycle is 1 time unit on both ECUs. The size of m_1 is 4 minislots.

Upper service using method 1: We now show the result of using β_1^u as obtained using method 1 in the computation of outgoing arrival curves. Let the arrival curve of m_1 be $\overline{\alpha}_1 = [\overline{\alpha}_1^u, \overline{\alpha}_1^l]$, which is periodic with period 21. Figure 5 shows the service curves β_1 available to m_1 (note that β_1^u is calculated using Method 1). The output arrival curve $\overline{\alpha}_1'$ (shown in Figure 6) of m_1 is computed by Equations 1 and 2 using $\overline{\alpha}_1$ and β_1. It can be seen from Figure 6 that $\overline{\alpha}_1^u$ is incorrect, as it says that in any time interval of less than 4 units, there can be maximum zero messages in the outgoing message stream, which is incorrect, because a message can appear at the output any time, i.e., in any interval of time 0^+, there can be a message at the output. Next, this $\overline{\alpha}_1'^u$ is used as the incoming arrival curve for task T_y. Using this $\overline{\alpha}_1'^u$ for task T_y, the remaining lower service after processing T_y is shown in Figure 7 which shows that the remaining lower service curve after processing T_y is also incorrect, because it shows that in any interval of 2 time units, minimum service is 2 cycles, which is not the case because in any interval of 2 time units, lower service is 0 as higher priority task T_y may be occupying the ECU. Thus, we have shown how the incorrect $\alpha_1'^u$ further introduces errors in the analysis.

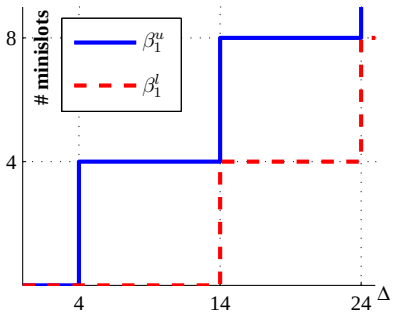


Figure 5: β_1 for the example

Upper service using method 2: If we use method 2 for obtaining the upper service curve β_1^u, then the upper service unutilized by m_1 (calculated using Eq. 3) comes out to be incorrect as shown in Figure 8. In reality, in any cycle, either the entire service available to a message is left unutilized or it is consumed entirely. But from Figure 8, we see that in the communication cycle in the interval 10 to 20, only 1 minislot is unutilized, and thus it is incorrect. Further, this incorrect service curve will lead to over optimistic analysis when it is used for obtaining the service available to m_2.

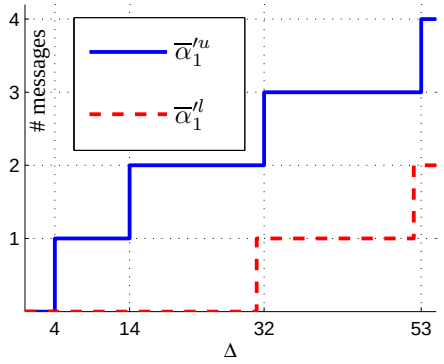


Figure 6: $\overline{\alpha}_1'$ for the example

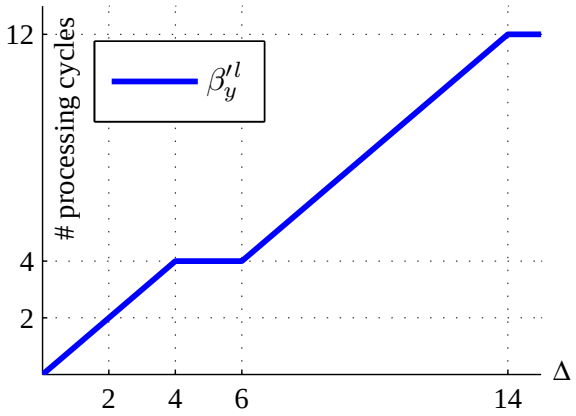


Figure 7: $\beta_y'^l$ for the example

Note: We defer to the next section the discussion to show that the lower service curve available to a message given by [5] results in pessimistic remaining lower service curve, and we also present a tighter lower service curve than that of [5].

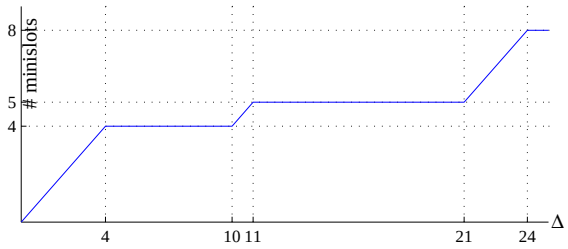


Figure 8: Upper service unutilized by m_1

5. REVISED FLEXRAY MODEL

In this section, we revise the FlexRay model given by [5] by giving the service curve available to a message.

In contrast to the previous work of [5] in which the messages are assumed to be of fixed sizes, we assume that m_i requires minimum of k_i^l and a maximum of k_i^u minislots.

We model the DYN message as shown in Figure 9. This

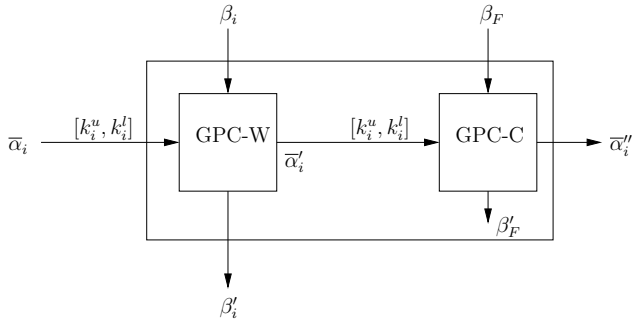
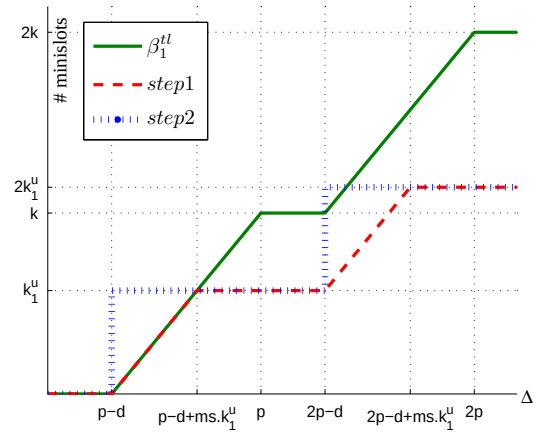


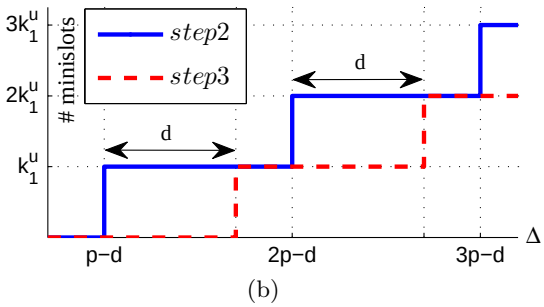
Figure 9: RTC model for message m_i

model is based upon the fact that the response time of a message consists of a waiting time and a communication time. The waiting time of a message is the time from its arrival till it gets access to the bus to start transmission. Figure 1 shows the waiting time for message m_3. The communication time (minislots_required/bus_speed) is the time the message takes for transmission once it occupies the bus.

Thus, there are two GPCs (GPC-W and GPC-C) in the model of any DYN message.



(a)



(b)

Figure 10: (a) Steps 1 and 2 and (b) Step 3 to obtain β_1^l according to our method

Thus, we need to modify the total service available to m_i i.e., $\beta_i^t = [\beta_i^{tu}, \beta_i^{tl}]$ to obtain $\beta_i = [\beta_i^u, \beta_i^l]$ in such a way that GPC-W represents only the waiting time of m_i. The communication time of m_i will be taken into account by GPC-C whose incoming service curve β_F is the "full service" [13] with the capacity equal to the bandwidth of the FlexRay bus.

For the highest priority message m_1, $\beta_1^t = \beta$, where β is the total service offered by the unloaded DYN segment.

To obtain β_i^l, we apply the following procedure.

1. Nullify the communication cycles of β_i^{tl} containing less than k_i^u minislots. Extract k_i^u minislots from the remaining communication cycles. The resulting curve is $\beta_{i,1}^{tl}$. See Figure 10(a).

2. For each increasing segment of the curve $\beta_{i,1}^{tl}$, discretize that segment at the point where it starts increasing. The resulting curve is $\beta_{i,2}^{tl}$. See Figure 10(a).

3. Shift the curve $\beta_{i,2}^{tl}$ by d time units to obtain β_i^l. See Figure 10(b).

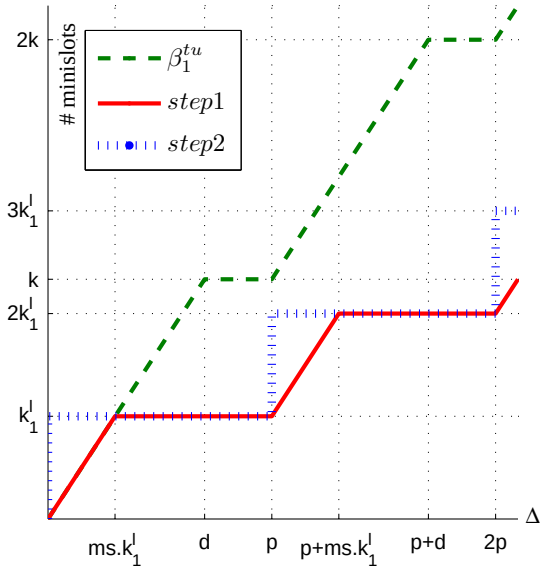


Figure 11: Steps 1 and 2 to obtain β_1^u according to our method

To obtain β_i^u, we apply the following procedure.

1. Nullify the communication cycles of β_i^{tu} containing less than k_i^l minislots. Extract k_i^l minislots from the remaining communication cycles. The resulting curve is $\beta_{i,1}^{tu}$. See Figure 11.

2. For each increasing segment of the curve $\beta_{i,1}^{tu}$, discretize that segment at the point where it starts increasing. The resulting curve is β_i^u. See Figure 11.

Now, the worst case response time of the message is calculated using Eq. 7 with the service curves β_i^l and β_F^l, and the arrival curve α_i.

The service unutilized by m_i, $\beta_i' = [\beta_i'^u, \beta_i'^l]$ is obtained using Eq. 3 and 4. But this service is specific to message m_i [5]. So we apply inverse of steps 2 (make the discrete curve

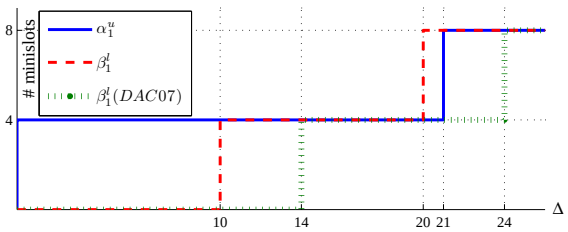


Figure 12: Upper arrival curve for m_1 and lower service curves (revised and that of DAC07)

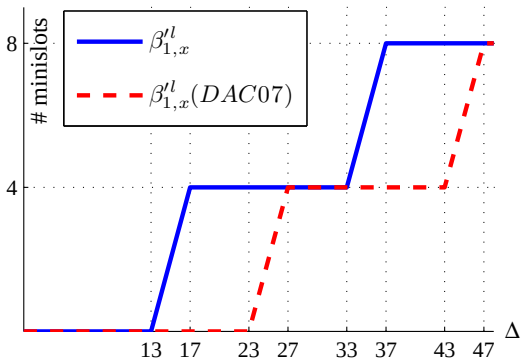


Figure 13: Comparison between unutilized service

sloped) and step 3 (shift the curve by d time units in the reverse direction) to β'^l_i to give $\beta'^l_{i,x}$. We apply inverse of step 2 (make the discrete curve sloped) to β'^u_i to give $\beta'^u_{i,x}$.

To take into account the property of one empty minislot, we subtract one minislot from each communication cycle of $\beta'_{i,x}$ (offering > 0 service) to give $\beta'_{i,y}$. (If the service is from Δ =13 to 17, then service would be from 14 to 17 for the lower curve and it would be from 13 to 16 for the upper curve).

The service unavailable to m_i [5] is $\beta^n_i = [\beta^{nu}_i, \beta^{nl}_i]$ where $\beta^{nu}_i(\Delta) = \sup_{0\leq\lambda\leq\Delta}\{\beta^{tu}_i(\lambda) - \beta^{tu}_{i,1}(\lambda)\}$ and $\beta^{nl}_i(\Delta) = \sup_{0\leq\lambda\leq\Delta}\{\beta^{tl}_i(\lambda) - \beta^{tl}_{i,1}(\lambda)\}$ (adjust one minislot from the cycles of β^t_i which could not offer required service).

Now the total service available to the next highest priority message m_{i+1} is $\beta^t_{i+1} = \beta^n_i + \beta'_{i,y}$.

Discussion on the lower service offered to the message: Again consider the system shown in Figure 4. Figure 12 shows the upper arrival curve for m_1 and the lower service curve which we derive and that of [5] (note that DAC07 in the figure indicates that the curve is obtained using the analysis of [5]). It can be seen from this figure that, when calculating $\beta'^l_1(21)$, the value of $\beta^l_1(20) - \alpha^u_1(20)$ will be 4, but $\beta^l_1(20)(DAC07) - \alpha^u_1(20)$ will be 0. This value of β'_1 will be used in computing $\beta'^l_{1,x}$. Figure 13 shows $\beta'^l_{1,x}$ (the service unutilized by m_1 after applying the inverse of step 2 and 3) and also its counterpart obtained using the model of [5]. It can be shown that m_1 can not be present during any two consecutive cycles. We can see from the Figure 13 that the unutilized service given by [5] is pessimistic as it shows that during any two consecutive cycles, m_1 could have been present.

6. CONCLUSIONS

In this paper, we revised the analysis of the FlexRay DYN segment given by [5]. The revised analysis also supports variable length messages. Our technique of separately considering "waiting time" and "transmission time" of the message can be used in the analysis of other similar protocols.

7. REFERENCES

[1] J.-Y. L. Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. LNCS 2050, Springer-Verlag, 2001.

[2] CAN specification, ver 2.0, Robert Bosch GmbH, 1991.

[3] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proc. 6th Design Automation and Test in Europe (DATE)*, pages 190–195. IEEE Press, 2003.

[4] FlexRay Communications System Protocol Specification, Ver 2.1, Revision A. Available from `http://www.flexray.com/`.

[5] A. Hagiescu, U. D. Bordoloi, S. Chakraborty, P. Sampath, P. V. V. Ganesan, and S. Ramesh. Performance analysis of FlexRay-based ECU networks. In *Proceedings of the 44th ACM/IEEE Design automation conference (DAC 2007)*, pages 284–289. ACM, 2007.

[6] W. Haid and L. Thiele. Complex task activation schemes in system level performance analysis. In *Proc. 5th Intl. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 173–178. ACM Press, 2007.

[7] M. P. J. Berwanger and R. Griessbach. A new high performance data bus system for safety-related applications, 2000. Available from `http://www.byteflight.de`.

[8] H. Kopetz and G. Bauer. The time-triggered architecture. *Proc IEEE*, 91(1):112 – 126, 2003.

[9] G. Leen and D. Heffernan. TTCAN: a new time-triggered controller area network. *Microprocessors and Microsystems*, 26(2):77–94, 2002.

[10] Local Interconnect Network Specification, LIN Consortium. Available from `http://www.lin-subbus.de`.

[11] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing analysis of the flexray communication protocol. *Real-Time Systems*, 39(1-3):205–235, 2008.

[12] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *International Symposium on Circuits and Systems (ISCAS 2000)*, volume 4, pages 101–104, Geneva, Switzerland, Mar. 2000.

[13] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse. System architecture evaluation using modular performance analysis - a case study. *Software Tools for Technology Transfer*, 8(6):649 – 667, Oct. 2006.