Generation of Communication Schedules for Multi-Mode Distributed Real-Time Applications

Akramul Azim, Gonzalo Carvajal, Rodolfo Pellizzoni, Sebastian Fischmeister
Department of Electrical and Computer Engineering
University of Waterloo, Ontario, Canada
{aazim, gcarvaja, rpellizz, sfischme}@uwaterloo.ca

***Abstract*—A key problem in designing multi-mode real-time systems is the generation of schedules to reduce the complexities of transforming the model semantics to code. Moreover, distributed multi-mode applications are prone to suffer from delays incurred during mode changes. We therefore aim to generate communication schedules that have low average mode-change delay for multi-mode real-time distributed applications.**

In this paper, we use optimization constraints associated to timing requirements to generate state-based schedules for multi-mode communication systems, and illustrate the workflow for generating schedules from specifications through a real-time video monitoring case-study. Our experiments in the case-study demonstrate that schedules generated using the proposed method reduce the average mode-change delay in relation to a randomized algorithm and the well-known EDF scheduling algorithm.

I. Introduction

Current trends in distributed systems are pushing the boundaries of existing real-time networks. Increased number and complexity of distributed devices, and integration of multiple real-time domains with traditional computers in a single network, are quickly turning legacy fieldbuses obsolete due to their limited bandwidth and incompatible protocols.

Increasing interest in Real-Time Ethernet (RTE), for example, provides strong evidence of this trend [1]. In recent years, both industry and academia have reported experimental evidence for hard real-time communication on top of Ethernet infrastructure [2], [3]. A common characteristic among the proposed solutions is the use of enhanced devices with specific modules for Time Division Multiple Access (TDMA) arbitration. To provide real-time guarantees, TDMA networks require careful planning of time-critical communication tasks, which must be scheduled and verified in advance. In practice, static TDMA schedules are complex to design, and lead to inefficient bandwidth utilization, since they must reserve time slots to handle the worst-case, even though it rarely occurs.

State-based scheduling is an alternative arbitration technique for hard real-time systems. State-based schedules allow developers to describe multiple operational modes, or states, and encode transitions from one state to another based on conditions that change at runtime. This property increases the flexibility, enable quick responses to changing operational conditions, and improve the bandwidth utilization compared to static TDMA configurations, while keeping the system analyzable and verifiable. Multiple case studies show the advantages of this approach in domains such as control theory [4], hybrid systems [5], hierarchical scheduling [6], and in general bursty demand models [7]. The Network Code framework [8] provides a complete development environment including an expressive language to describe TDMA schedules with conditional branching, tools to verify the schedules before runtime, and a powerful hardware-accelerated platform to deploy and test solutions in practical scenarios [9]. However, a big limitation of this approach is that developers must still write the program at low-level, requiring a good understanding of the underlying technology and runtime environment, and perform fine tuning of configuration parameters to fit particular application requirements. This approach is time-consuming, prone to errors, and inadequate for medium/large-scale systems, specially since conditional transitions add a new level of complexity to the schedule design and verification.

978-3-9815370-2-4/DATE14/©2014 EDAA

Generating real-time schedules for efficient utilization of the available bandwidth, with data dependencies and conditional execution is a challenging and relevant problem for next-generation distributed systems. This work proposes a novel methodology to generate state-based communication schedules from a high-level specification of the distributed components, moving away the complexity of schedule design from the developer. Starting from a state machine description of the distributed tasks, the proposed workflow generates abstract representations of state-based schedules for any feasible system, which can then be mapped to executable entities. This paper walks through the individual steps using an example case-study based on real-time video streaming over an Ethernet network, which is a recurrent application in the automotive domain. The results demonstrate that the generated schedules meet the real-time guarantees, while minimizing the average delay to switch from one operational mode to another with respect to a set of valid schedules generated using both Earliest-Deadline-First (EDF) and random mapping of messages to time slots. The analysis from the particular case-study can be easily generalized to any network based on TDMA.

The remainder of the paper is structured as follows: Section II revises the concept of state-based scheduling, and introduces relevant terminology for the schedule generation workflow. Section III illustrates the steps for the generation of state-based schedules from component-level specifications using a video streaming case-study. Section IV provides a brief overview of related work, and finally Section V concludes the paper.

II. Overview of State-Based Scheduling

This section reviews the concept and introduces formal definitions of state-based scheduling.

A. State-Based Schedules

A state-based schedule is an abstract representation of communication systems based on TDMA with on-the-fly

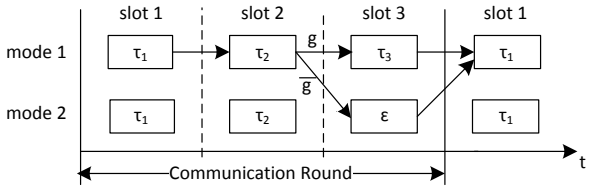


Fig. 1: State-based representation of a TMR application

decisions. A TDMA communication system consists of a set of stations that exchange messages through a broadcast network.

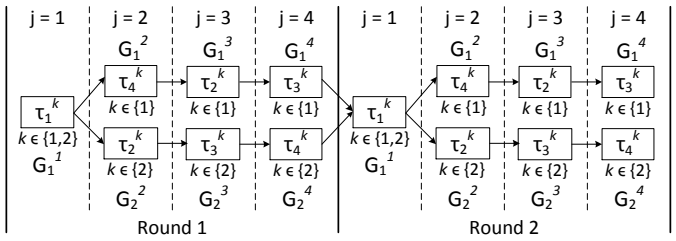
TDMA schedules divide time into non-overlapped slots and rounds. Each slot is defined by a start point and a slot length. A scheduler performs a message-to-slot mapping based on the timing requirements for the system. A *linear schedule* maps a slot to either only one message or leaves it empty. A communication round refers to a sequence of messages that repeats endlessly as the system executes.

An *operational mode* describes a system state and specifies the messages that the system needs to communicate when running on the associated mode. Each operational mode is associated to a predefined linear schedule that meets the timing requirements for the messages. State-based schedules encode a set of linear schedules associated to different modes using a global-time base. As a result, different messages from different modes can be mapped to the same slot, but only one mode can be active at any slot during runtime. The schedules can encode guarded transitions that allow the system to change from one mode to another between consecutive slots within a single communication round.

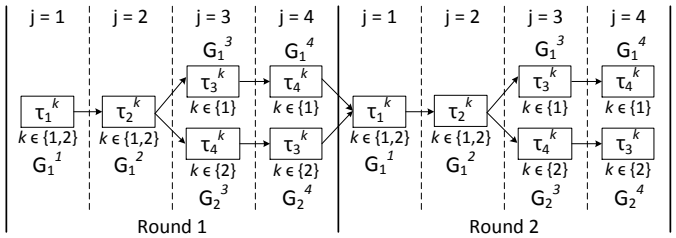
To illustrate the concepts, let us consider an example of a Triple Module Redundancy (TMR) application. In a typical setup for TMR, three sensors transmit independent samples of the same variable in consecutive messages τ_1, τ_2, and τ_3. A voting controller receives these messages and performs a majority vote to determine the final value. On the one hand, in a static TDMA configuration, the sequence of transmitted messages is determined only by the progression of time, and then the stations will always transmit the three messages, even if τ_1 and τ_2 are already decisive for the voting. On the other hand, a state-based schedule can perform a preliminary voting after receiving the first two samples, and if the voting is already decisive, then the slot associated to the third sample can be empty, leaving the medium available for other purposes such as best-effort traffic. Fig. 1 illustrates this behavior. The system can operate in two modes: $mode\ 1$ considers that the system needs to communicate the three messages, and $mode\ 2$ considers that the vote is already decisive after the second message. The system starts in $mode\ 1$, and after transmitting τ_2 it checks the guard $g : \tau_1 \neq \tau_2$ to decide whether to transmit a message or leave the slot empty (ϵ) and available for other messages (such as best-effort data). After the third slot, a new round starts and the system resets to $mode\ 1$.

B. Formal Definitions

Let us consider a set of messages $\mathrm{T} = [\tau_1, \ldots, \tau_N]$. Each message τ_i has an associated period p_i, transmission time e_i, and implicit deadline $d_i = p_i$, all represented as entire multiples of an atomic time unit γ. Let us also consider a set $V = [v_1, \ldots, v_M]$ representing the operational modes, each one associated to a linear schedule. Considering a slot length of



(a) Overlapped message assignments in slot 1 form group G_1^1



(b) An alternative schedule with an additional overlap in slot 2

Fig. 2: Different valid state-based schedules for a particular system

γ, the communication round will have $\Gamma = \mathrm{LCM}(p_1, \ldots, p_N)$ slots, where Γ is the Least-Common-Multiple of periods of all the messages in T, i.e., the *hyperperiod* of the system. We say that the state-based schedule has an *overlap* in slot $j = 1, \ldots, \Gamma$ when different modes map the same message to that slot. Overlapped messages at slot j can be combined into a *group* G_m^j, where $1 \leq m \leq |V|$. A state-based schedule is a tree-like structure where each branch represents a *transition* from a group in slot j to one or more groups in slot $j + 1$.

A valid state-based schedule is one that meets the timing requirements for each linear schedule and the possible transitions between them. A particular system can have multiple valid schedules, and then different number of overlaps and groups. Fig. 2 shows an example of two different schedules for a system with two modes and $\Gamma = 4$. For illustrative purposes, let us assume that both schedules are valid for the particular system. The notation τ_i^k indicates that mode k maps τ_i to the corresponding slot. The schedule in Fig. 2a has an overlap for τ_1 in $j = 1$, which can be represented in group G_1^1. Fig. 2b shows an alternative schedule with an additional overlap for τ_2 in slot 2, which is represented in G_1^2.

A mode-change in a state-based schedule is a timed event that triggers a guarded transition to move from an old mode $v_s \in G_m^j$ to a new mode v_d in the next slot at the cost of a mode-change delay $\delta(v_s, v_d)$ defined as:

$$\delta(v_s, v_d) = \begin{cases} 0 & \text{if } v_d \in G_m^j \\ \Gamma - j & \text{otherwise} \end{cases} \tag{1}$$

where $\Gamma - j$ represents the time until the next communication round starts. The mode-change can occur immediately if a transition exists between mode v_s and v_d. If not, the mode-change occurs when a new communication round starts. Reducing the number of transitions decreases the number of groups which lowers the average mode-change delay (Theorem 1).

Theorem 1. *Given a set of valid state-based schedules for a particular system, the schedule with the fewest number of groups*

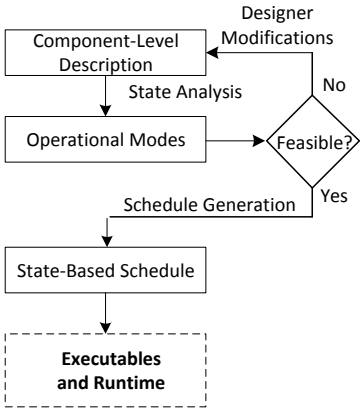


Fig. 3: Proposed Workflow

in a communication round will minimize the average mode-change delay $\overline{\delta}$ for a uniform probability of mode-changes.

Proof: Our proof is based on contradiction. Let us consider two valid schedules S and S' for a particular system, with S' having fewer number of groups in a communication round. Let us assume that $\overline{\delta}_{S'} > \overline{\delta}_S$. The value $|V - G^j_m|$ represents the number of modes that do not belong to G^j_m. Since a maximum number of $|G^j_m|$ modes can request a mode-change to the modes that do not belong to G^j_m, the total mode-change delay Δ for a uniform possibility of mode changes at the end of slot j for a group m and a transition is:

$$\Delta = \sum_{v_s, v_d \in V} \delta(v_s, v_d) = (\Gamma - j)(|V - G^j_m|)|G^j_m|$$

Since the system executes all the transitions with uniform probabilities, then the average mode-change delay $\overline{\Delta}$ for all slots for all possible groups at runtime is:

$$\overline{\Delta} = \frac{\sum_j \sum_m (\Gamma - j)(|V - G^j_m|)|G^j_m|}{\Gamma |\bigcup_j (\bigcup_m G^j_m)|}.$$

The average delay in the state-based schedule $\overline{\delta}_{S'}$ for a uniform possibility of mode changes is less than $\overline{\delta}_S$, because S' has fewer groups than that in S and therefore the value $|V - G^j_m|$ is less than that of in S. This contradicts that the average mode-change delay in S is lower than the average mode-change delay in S'. ■

This following section describes a workflow for schedule generation that uses the concept of groups to address the problem of message-to-slot mapping for state-based schedules, such that the resulting valid schedule minimizes the average mode-change delay.

III. Schedule Generation Workflow

This section walks through an example case-study to illustrate the necessary steps to generate state-based schedules from a component-level description.

A. Workflow Overview

Fig. 3 illustrates the steps for the proposed workflow to generate state-based schedules. The designer specifies a component-level description consisting of state machine descriptions and timing requirements for each task generating messages. Given that the system is correctly described and feasible, the proposed workflow will generate a valid schedule that minimizes the average mode-change delay.

The first step in the workflow is an analysis to combine the states of the individual components and obtain the operational modes of the system. This step also considers a feasibility test to verify that it is possible to meet the timing requirements for all the operational modes. If any of the operational mode is unfeasible (e.g., due to bandwidth limitations), then the designer must modify the system specifications. For feasible systems, the next step is the generation of a valid schedule that minimizes the average mode-change delay.

The previous steps generate an abstract representation describing the message-to-slot mapping for the system. These descriptions can then be translated into a programming language to generate executable abstractions. Previous work illustrates this process by mapping schedules designed by hand for simple applications to the Network Code framework, which offers a domain-specific programming language with conditional branching capabilities and a powerful hardware environment for real-time communication over Ethernet [9].

The rest of this section provides details for each step in the workflow using an example case-study based on the demonstration setup described in [9]. The application considers the real-time streaming of multiple video sources on top of Ethernet, which can change the resolution according to specific operational conditions for efficient utilization of the bandwidth.

B. Application Example and Assumptions

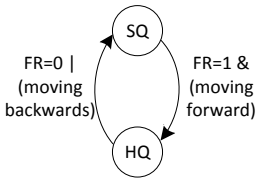
Let us consider an embedded video monitoring system for mining trucks. Drivers need some kind of monitoring system for increased security when sharing the road with smaller vehicles and people because of the dimensions of these trucks (typically over 7 mt. high). The system uses four video cameras, each one transmitting a stream of the surroundings of each wheel to displays located in the driver's cabin. All devices connect through a real-time capable Ethernet network operating at 1[Gbit/s] [9].

The cameras operate in two states that differ in the number of Frames-Per-Second (FPS) : Standard Quality (SQ) and High Quality (HQ). The cameras operate in SQ by default. Each wheel includes a sensor that detects proximity to surrounding objects. When a sensor activates, the cameras switch to HQ. An additional condition is that the sensors activate according to the movement of the truck: front sensors only activate if the truck is moving forward, and rear sensors only activate if the truck is moving backwards.

For the analysis we only consider the scheduling of the video streams, and assume that all distributed components are synchronized to a global clock reference. In practice, the model specification must also consider the scheduling of periodical sensor readings and synchronization messages, which require much less bandwidth than the video data [9].

C. Component-Level Description

Fig. 4 shows a state machine representation and transition table for the camera attached to the front right wheel. The inputs



(a) State Machine

FCR, FCL, RCR, RCL	s_k	s_{k+1}
1, 0, 0, 0	X	HQ
1, 1, 0, 0	X	HQ
others	X	SQ

(b) Transition Table

Fig. 4: Representation of the states for each camera

Front Right (FR), Front Left (FL), Rear Right (RR), and Rear Left (RL) represent the status of the sensors attached to each wheel; and s_k and s_{k+1} represent the current and next state, respectively. The designer must provide a similar representation for each camera, together with the timing requirements for each state (addressed in Section III-D).

Alternatively, the designer can use high-level languages such as AADL [10] or UML-MARTE [11] to specify the system behavior, and use a parser to extract the transition tables and timing specifications.

D. State Analysis

The algorithm presented in [12] allows us to compute the operational modes of the system as the cross products of the state machines for the individual components. In the case-study, the condition that front and rear cameras cannot operate at HQ at the same time leaves only seven possible operational modes out of the sixteen possible.

Table I shows the parameters of interest to perform feasibility analysis for the case-study. These parameters must be encoded together with the transition tables. The nominal transmission time represents the time required to transmit a single video frame of 640x480 pixels, with a pixel-width of 32 bits, over a 1[Gbit/s] link. Since the maximum payload for a standard Ethernet frame is 1500 bytes, video frames are transmitted as a sequence of Ethernet frames. The reported time accounts for the overhead related to Ethernet headers and Inter-Frame Gap (IFG). For simplicity, we omit the propagation latency and additional processing in the path between cameras and displays, which will depend on the physical configuration of the network. In practice, designers must provide a worst-case value for these parameters and consider them in the total transmission time for each message. The table also shows the periods associated to different FPS. Considering an atomic unit for the schedule equal to the minimum transmission time for a message, we normalize the timing specifications to this time unit, and floor the normalized period. This processing over-estimate the actual requirements, but allows us to represent all the timing as multiple integers of the time unit.

For the system to be feasible, the total channel utilization for each operational mode cannot be greater than the available bandwidth for scheduled traffic. This is:

$$U(v_k) = \sum_{\tau_i \in v_k} \frac{e_i}{p_i} \leq \frac{B}{L} \tag{2}$$

where B is the bandwidth assigned to scheduled messages, and L is the link capacity, with $B \leq L$.

Table II summarizes the utilization test for all operational mode when setting the SQ mode to 15[FPS], and the HQ mode to either 30[FPS] or 60[FPS]. Considering that all link capacity is available for the video streams, the feasibility test will be $U(v_k) \leq 1$. We see if the HQ mode is set to 60[FPS], there are two modes that fail the feasibility test (bold numbers), and then the designer must either need to change the specifications or modify the system. In this case, reducing the quality of the HQ mode to 30[FPS] makes the system feasible.

TABLE I: Timing Requirements for Different Video Qualities

FPS	Nominal [μs]		Normalized	
	Trans. time	Period	Exec. time	Period
15	7600	66700	1	8
30	7600	33350	1	4
60	7600	16720	1	2

TABLE II: Feasible modes with messages specifications

ID	Mode	Utilization (SQ=15[FPS])	
	(FCR, FCL, RCR, RCL)	HQ=30[FPS]	HQ= 60[FPS]
1	(SQ,SQ,SQ,SQ)	0.5	0.5
2	(SQ,SQ,SQ,HQ)	0.625	0.875
3	(SQ,SQ,HQ,SQ)	0.625	0.875
4	(SQ,SQ,HQ,HQ)	0.75	**1.25**
5	(SQ,HQ,SQ,SQ)	0.625	0.875
6	(HQ,SQ,SQ,SQ)	0.625	0.875
7	(HQ,HQ,SQ,SQ)	0.75	**1.25**

E. Schedule Generation

To find an optimal state-based schedule with respect to minimizing the number of groups, this work uses integer linear programming (ILP) to find optimal assignments of messages to slots. Minimizing the number of groups provides the optimal assignments of messages to slots, which are based on the constraints on timing requirements of messages of each mode and the characteristics of state-based schedules.

In addition to the parameters of the system model, the ILP model uses some variables to find the optimal assignments of messages to slots. The variable α_i represents the number of instances for every τ_i until the hyperperiod such that $\alpha_i = \frac{\Gamma}{p_i}$. The variable x_{ij}^k represents the usage of a time slot for $i \in N$, $j \in \{1, \ldots, \Gamma\}$ and $k \in V$. Therefore,

$$x_{ij}^k = \begin{cases} 1 & \text{if message } \tau_i \text{ is allocated to slot } i \text{ in mode } k \\ 0 & \text{otherwise.} \end{cases}$$

The number of groups in a state-based schedule will increase if different messages of different modes are allocated to the same slot and the previous slots. The variable s_{ij}^k is used to reduce the possibilities of such assignments of messages to slots. The value of s_{ij}^k is set to 1 if a message i of mode k is allocated to a slot j, and there exists not only a message except i of any other modes that is allocated to the same slot j but also a different message than i of any other modes allocated to any of the previous slots (i.e., $1, \ldots, j-1$). This is required because the increase in number of groups not only depend on allocating different messages of other modes in the current slot but also in the earlier time slots due to increasing the number of transitions. Therefore,

$$s_{ij}^k = \begin{cases} 1 & \text{if } x_{ij}^k = 1 \wedge \\ & \exists x_{uj}^v, st: u \neq i \wedge v \neq k \wedge x_{uj}^v = 1 \wedge \\ & \exists x_{aq}^b, st: a \neq i \wedge b \neq k \wedge \\ & \quad q \in \{1, \ldots, j-1\} \wedge x_{aq}^b = 1 \\ 0 & \text{otherwise.} \end{cases}$$

The variable w_{ij}^k is used to determine whether different messages of other modes are allocated to the same slot j if a message i of mode k is already allocated to slot j. The variable z_{ij}^k is used to determine whether different messages of other modes are allocated to the previous slots if a message i of mode k is allocated to slot j. The ILP model shown below minimizes the number of groups.

$$\begin{aligned} \min \quad & \sum_{\forall \mathrm{i} \in \mathrm{N}, \mathrm{j} \in \{1,\ldots,\Gamma\}, \mathrm{k} \in \mathrm{V}} s_{ij}^k. \\ st. \quad & \mathbf{C}_1\{\forall \mathrm{i} \in \mathrm{N}, \mathrm{k} \in \mathrm{V}\}: \\ & \sum_{j=gp_i^k+1}^{gp_i^k+p_i^k} x_{ij}^k \geq c_i^k \ , g = 0, \ldots \ \alpha_i - 1; \\ & \mathbf{C}_2\{\forall j \in \Gamma, k \in \mathrm{V}\}: \\ & \sum x_{ij}^k \leq 1, \\ & \mathbf{C}_3\{\forall \mathrm{i} \in \mathrm{N}, \mathrm{j} \in \Gamma, \\ & \mathrm{k} \in \mathrm{V}, \mathrm{u} \in \mathrm{N} - \{i\}, \mathrm{v} \in \mathrm{V} - \{\mathrm{k}\}: \\ & w_{ij}^k \geq x_{uj}^v, \\ & \mathbf{C}_4\{\forall \mathrm{i} \in \mathrm{N}, \mathrm{j} \in \Gamma, \\ & \mathrm{k} \in \mathrm{V}, \mathrm{a} \in \mathrm{N} - \{i\}, \mathrm{b} \in \mathrm{V} - \{\mathrm{k}\}, \\ & \mathrm{q} \in \Gamma - \{\mathrm{j} \ldots \Gamma\}: \\ & z_{ij}^k \geq x_{aq}^b, \\ & \mathbf{C}_5\{\forall \mathrm{i} \in \mathrm{N}, \mathrm{j} \in \Gamma, \mathrm{k} \in \mathrm{V}\}: \\ & s_{ij}^k \geq x_{ij}^k + w_{ij}^k + z_{ij}^k - 1, \\ & \mathbf{C}_6\{\forall \mathrm{i} \in \mathrm{N}, \mathrm{j} \in \Gamma, \mathrm{k} \in \mathrm{V}\}: \\ & s_{ij}^k \geq 0, \end{aligned}$$

In the ILP model, a number of constraints represent the characteristics of the optimized schedule. Constraint C_1 specifies that all messages at least get the computation units in their periods. Constraint C_2 and C_6 are bounds to guarantee x_{ij}^k and s_{ij}^k being binary. Constraint C_3, C_4, and C_5 compute s_{ij}^k. These constraints manage to find optimal assignments of x_{ij}^k that the decomposition method (Def. 1) uses to construct a state-based schedule.

Definition 1 (Decomposition method)**.** *Given all schedulable and reachable groups at any time slot, a decomposition method derives all groups in the next time slot from x_{ij}^k followed by a labelled transition (i.e., guard).*

Using the decomposition method as defined above, it is possible to generate an optimized schedule either based on provisioning empty slots to transmit best-effort traffic or allowing messages to execute more than their timing requirements. In the latter, the schedule has fewer groups and therefore faster average mode-change delay, but lower bandwidth for best-effort traffic. In this paper, we focus on maximizing the available bandwidth for best-effort traffic to gain the full advantage of using state-based schedules. According to Def. 1, we define a method (not shown due to space limitations) to construct a state-based schedule from the values of x_{ij}^k based on provisioning empty slots. Using x_{ij}^k, the method finds all groups from $j = 1$ onwards, along the slots, up to the slot Γ, where a group contains modes that either have overlapped messages or nothing (ϵ) to transmit. A group G initially contains all messages and uses the decomposition method to create groups G_{ij}^k. Fig. 5 shows the state-based schedule for the case-study prioritizing best-effort traffic.

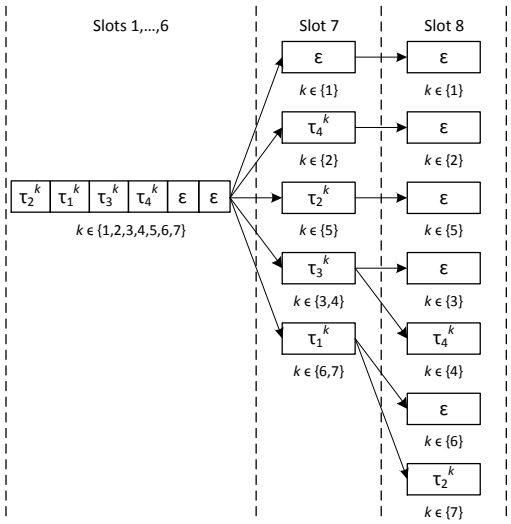


Fig. 5: An optimized best-effort oriented schedule using decomposition

To demonstrate that the solution to the described optimization problem assigns the messages to slots efficiently, we use random assignments of x_{ij}^k that also meet the timing requirements of messages of each mode. Fig. 6 shows the difference in the normalized number of transitions between the 10000 randomized schedules and the optimized schedule for a hyperperiod of 4, 8, 12, 16, 20, 24, 28, and 32 time slots using the case-study through changing the period of the SQ message. The error bars in the randomized schedules represent the normalized maximum and minimum number of transitions among 10000 schedules. We also generate state-based schedules using the assignments of messages to slots through EDF scheduling policy (Fig. 6). The geometric mean of the normalized number of transitions for the schedules using optimized, randomized, and EDF x_{ij}^k assignments are $0.6094, 0.9505$, and 1 respectively. The average mode-change delay for the generated schedule using optimized x_{ij}^k is also significantly less (Fig. 7) than the delay in the schedules generated using either randomized or EDF-based x_{ij}^k.

We implemented a set of open-source scripts for each step of the workflow. Users can download these scripts [13] and verify the results for the case-study included as an example, or start with new system specifications. The current version of the workflow scripts are based on Matlab and use the AMPL/GUROBI optimization problem solver.

IV. Related Work

Traditional real-time networking protocols allow limited control to the applications over the communication behaviour at runtime. For example, developers must assign message priorities statically on a CAN bus to ensure that the priorities

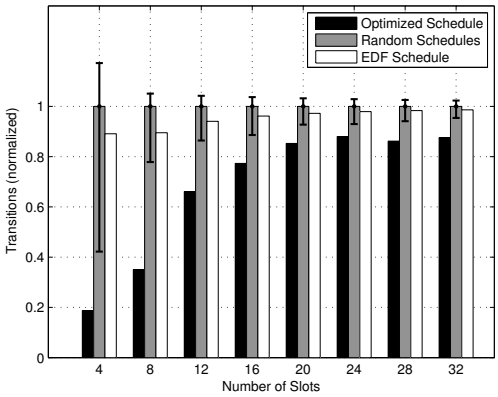


Fig. 6: Normalized number of transitions in generated schedules using randomized, EDF-based, and optimized x_{ij}^k assignments

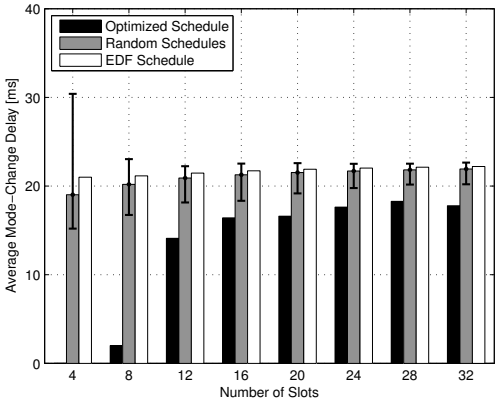


Fig. 7: Average mode-change delay in generated schedules using randomized, EDF-based, and optimized x_{ij}^k assignments

are unique [14]. FlexRay [15] follows a static TDMA approach with a specific slot for dynamic traffic at the end of each round. Stations must wait for that specific slot to transmit dynamic messages, and its timing is not guaranteed.

Some recent work explore the concepts of state-based schedules, but they lack any generation technique of schedules from high-level specifications. For example, in [4], the authors demonstrate the advantage of state-based scheduling for control systems, but using a small scale system, and the schedules are not necessarily optimized. Moreover, the work uses timed automata [16] to express the schedule using regular expressions. This work uses the notion of slots and communication rounds to reduce complexity for analysis and verification. In [17], the authors propose mechanisms to synthesize clocked graphs [18] with the Network Code framework, but avoids generation and optimization of state-based schedules.

V. Conclusion

Model-driven development is complex for time-critical systems not only because of meeting deadlines at runtime, but also how effective the generated schedules that represent the runtime behaviour. A well understood approach of model-driven development is to create a system-level design and generate schedules specific to the execution framework. In this paper, we present a workflow with an illustration of a real-time video streaming case-study to implement higher level abstractions through generating state-based schedules that facilitate conditional executions at runtime. We also demonstrate the generated schedules using constraints through an optimization solver are better than schedules generated using EDF and a randomized algorithm because of lower average mode-change delays.

Acknowledgments

The authors would like to thank Juan Rodriguez for early contributions to this work. This research was supported in part by projects NSERC DG 357121-2008, ORF-RE03-045, ORF-RE04-036, ORF-RE04-039, APCPJ 386797-09, CFI 20314, CMC Microsystems, and the industrial partners associated.

References

[1] J.-D. Decotignie, "The Many Faces of Industrial Ethernet," *IEEE Industrial Electronics Magazine*, vol. 3, no. 1, pp. 8–19, 2009.

[2] "Time-Triggered Ethernet," http://www.tttech.com.

[3] "Profinet," http://www.profibus.com.

[4] G. Weiss, S. Fischmeister, M. Anand, and R. Alur, "Specification and Analysis of Network Resource Requirements of Control Systems," in *Proc. of the Int. Conference on Hybrid Systems: Computation and Control (HSCC)*, 2009, pp. 381–395.

[5] M. Anand, S. Fischmeister, Y. Hur, J. Kim, and I. Lee, "Generating Reliable Code from Hybrid Systems Models," *IEEE Trans. on Computers*, vol. 59, no. 9, pp. 1281–1294, 2010.

[6] A. Easwaran, M. Anand, and I. Lee, "Compositional Analysis Framework Using EDP Resource Models," in *Proc. of the 28th IEEE Real-Time Systems Symposium (RTSS)*, 2007, pp. 129–138.

[7] L. T. X. Phan, S. Chakraborty, and P. S. Thiagarajan, "A Multi-Mode Real-Time Calculus," in *Proc. of the 29th IEEE Real-Time Systems Symposium (RTSS)*, 2008, pp. 59–69.

[8] S. Fischmeister, R. Trausmuth, and I. Lee, "Hardware Acceleration for Conditional State-Based Communication Scheduling on Real-Time Ethernet," *IEEE Trans. on Industrial Informatics*, vol. 5, no. 3, pp. 325–327, 2009.

[9] G. Carvajal, M. Figueroa, R. Trausmuth, and S. Fischmeister, "Atacama: An Open FPGA-based Platform for Mixed-Criticality Communication in Multi-segmented Ethernet Networks," in *Proc. of the 21st IEEE Int. Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2013, pp. 121–128.

[10] G. Lasnier, T. Robert, L. Pautet, and F. Kordon, "Behavioral Modular Description of Fault-tolerant Distributed Systems with AADL Behavioral Annex," in *Conference on New Technologies of Distributed Systems*, 2010, pp. 17–24.

[11] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, and J.-P. Diguet, "A co-design approach for embedded system modeling and code generation with uml and marte," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2009, pp. 226–231.

[12] S. C. Hsieh, "Product Construction of Finite-State Machines," in *Proc. of the World Congress on Engineering and Computer Science*, 2010, pp. 141–143.

[13] "Open-source scripts," http://www.mathworks.com/matlabcentral/fileexchange/44716.

[14] *CAN Specification, Version 2*, Robert Bosch GmbH, 1991.

[15] *FlexRay Communications System – Protocol Specification, Version 2*, FlexRay Consortium, 2004.

[16] R. Alur and G. Weiss, "Regular Specifications of Resource Requirements for Embedded Control Software," in *Proc. of the Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2008, pp. 159–168.

[17] D. Potop-Butucaru, A. Azim, and S. Fischmeister, "Semantics-Preserving Implementation of Synchronous Specifications Over Dynamic TDMA Distributed Architectures," in *Proc. of the International Conference on Embedded Software (EMSOFT)*, 2010, pp. 199–208.

[18] D. Potop-Butucaru, R. Simone, Y. Sorel, and J. Talpin, "Clock-Driven Distributed Real-Time Implementation of Endochronous Synchronous Programs," in *Proc. of the International Conference on Embedded Software (EMSOFT)*, 2009, pp. 147–156.