Holistic Scheduling of Real-time Applications in Time-Triggered in-Vehicle Networks

Menglan Hu, Jun Luo, *Member, IEEE*, Yang Wang, Martin Lukasiewycz, *Member, IEEE*, and Zeng Zeng

***Abstract*—As time-triggered communication protocols (e.g., TTCAN, TTP, and FlexRay) are widely used on vehicles, the scheduling of tasks and messages on in-vehicle networks becomes a critical issue for offering quality-of-service (QoS) guarantees to time-critical applications on vehicles. This paper studies a holistic scheduling problem for handling real-time applications in time-triggered in-vehicle networks where practical aspects in system design and integration are captured. The contributions of this paper are multi-fold. Firstly, this paper designs a novel scheduling algorithm, referred to as *Unfixed Start Time* (UST) algorithm, which schedules tasks and messages in a flexible way to enhance schedulability. In addition, to tolerate assignment conflicts and further improve schedulability, this paper proposes two rescheduling and backtracking methods, namely *Rescheduling with Offset Modification* (ROM) and *Backtracking and Priority Promotion* (BPP) procedures. Extensive performance evaluation studies are conducted to quantify the performance of the proposed algorithm under a variety of scenarios.**

***Index Terms*—Automotive electronics, distributed embedded systems, FlexRay, in-vehicle networks, list scheduling, real-time scheduling, task graphs, time-triggered systems.**

I. Introduction

As electronic and computer technologies rapidly evolve, today's cars are becoming complicated distributed embedded systems where various electronic devices such as controllers, sensors, and actuators are integrated to replace mechanical components. These electronic control units (ECU) require information exchange among each other via in-vehicle networks to support the execution of their tasks. For example, in today's luxury cars up to 70 ECUs exchange up to 2500 signals [1]. Data exchange on in-vehicle networks are implemented via communication protocols. One widely deployed protocol is the Controller Area Network (CAN) [2]. However, this event-triggered protocol is improper for novel real-time applications requiring predictable and robust communication. Hence, the design paradigm in the automotive industry is shifting from event-triggered systems (e.g., CAN) to time-triggered systems, which are based on time-triggered protocols such as Time-triggered Protocol (TTP) [5] and Time-triggered CAN (TTCAN) [3]. As the number of ECUs and functions in cars continues to increase, the needs of large-scale data exchanges for novel applications such as X-by-wire [6] has motivated the development of the FlexRay protocol [4], which is expected to be the de-facto standard in the automotive industry and has been deployed in new cars including new BMW-7 series.

Due to the wide deployment of the time-triggered protocols (e.g., TTCAN, TTP, and FlexRay) on vehicles [7], [8], [9], the scheduling of applications on such systems becomes a critical issue for offering quality-of-service (QoS) guarantees to time-critical applications on in-vehicle networks. Here, an application is referred to a set of tasks with timing constraints to perform a well-defined function in the vehicle. Each task running in a specified ECU is triggered by messages received from other tasks and sends messages to its downstream tasks. Consequently, the application can be abstracted as a directed acyclic graph (i.e., task graph) where nodes represents either the tasks or the messages and edges specify the precedences among the nodes. For instance, the in-cycle control in a vehicle is triggered when the messages of 4 wheel positions are available (broadcast by the corresponding tasks of the wheels) and thereby updates its control outputs (tasks). Given that such applications are regularly repeated, the scheduling problem can be formulated as a periodic task graph scheduling problem.

A number of studies for scheduling in time-triggered systems have been reported. Many existing works mainly focused on scheduling messages while tasks were neglected [28], [30], [29], [12]. The isolated message scheduling may severely limit the overall performance and feasibility of all applications which consist of both tasks and messages. A handful of studies have studied the holistic scheduling on both tasks and messages on time-triggered systems [20], [13], [18], [19], [21]. However, most of the above mentioned studies relied on mathematical programming techniques, which cannot scale to large-scale systems. Due to the sharply increasing amount of software functionality in modern vehicles, efficient and scalable heuristic algorithms are desired. Over the past decades, many heuristic algorithms for scheduling task graphs in multiprocessors have been proposed in literature [14], [15], [16], [26], [25], [24]. However, these heuristics were not designed for time-triggered in-vehicle networks. Also, most of the works even neglected the contention on communication

This work was supported in part by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) programme.

M. Hu is with the School of Computer Engineering, Nanyang Technological University, Singapore and also with the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, Wuhan, China. This paper was done when he worked in NTU. E-mail: humenglan@gmail.com.

J. Luo is with the School of Computer Engineering, Nanyang Technological University. E-mail: junluo@ntu.edu.sg.

Y. Wang is with the Faculty of Computer Science, University of New Brunswick, Fredetricton, Canada. E-mail: ywang8@unb.ca.

M. Lukasiewycz is with TUM CREATE Centre, Singapore. E-mail: martin.lucasiewycz@tum-create.edu.sg.

Z. Zeng is with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore. E-mail: elezengz@nus.edu.sg.

resources.

In addition, this paper considers practical needs in system integration in automotive industries. In practice, applications in vehicles are developed and tested by different teams and are integrated in a later stage. For example, BMW may integrate various sub-systems (i.e., partial schedules of various software functionality) offered by its suppliers onto its cars. The partial schedules have been verified by the suppliers so that possibly the end-to-end delay of a task graph is constrained by both lower bound and upper bound while in traditional models end-to-end delays are only bounded by upper bounds (i.e., deadlines). Further, possibly the end-to-end delay in a partial schedule is even fixed by the suppliers and cannot be changed; otherwise applications need be tested again. Accordingly, complex timing constraints are associated with the applications in system integration, enhancing the difficulty in schedulability.

Given the practical demands in system design and integration for time-triggered vehicle-carried systems, the previous approaches may not be directly applied and are likely to suffer from severe performance deterioration even if they may be revised to adapt for the novel needs. To this end, this paper contributes a set of novel and scalable heuristic algorithms.

Firstly, this paper proposes a novel scheduling algorithm, referred to as *Unfixed Start Time* (UST), which flexibly schedules tasks and messages such that their actual start times are not fixed during the scheduling. The flexibility offered by the algorithm is a significant advantage when compared to previous list scheduling heuristics (e.g., [26], [24], [14]). Also, the contention on communication resources in time-triggered systems is explicitly addressed in UST.

In addition, to tolerate assignment conflictions brought by complex and hard timing constraints and further improve schedulability, this paper proposes two rescheduling and backtracking approaches, namely *Rescheduling with Offset Modification* (ROM) procedure and *Backtracking and Priority Promotion* (BPP) procedure. Upon a conflict in time allocation, ROM reschedules the conflicted application with an adjusted offset (i.e., release time) such that the scheduling of different applications can be staggered to resolve conflicts. Once ROM is not helpful, BPP backtracks a number of previously scheduled applications to create space for the conflicted application and reschedules remaining applications with the priority promoted for the conflicted application.

The remainder of this paper is organized as follows. Section II discusses the related work. Section III introduces mathematical models, assumptions, and problem formulation. Section IV describes the proposed algorithms in great detail. Section V presents simulation results, with conclusions following in Section VI.

II. Related Work

A number of studies on scheduling in time-triggered in-vehicle networks have been reported in literature. Park et al. [10] proposed a FlexRay network parameter optimization method which can determine the lengths of the static slot and the communication cycle. In [30] and [28], the message scheduling problems were solved via nonlinear integer programming (NIP) for static segment and dynamic segment, respectively. In particular, [30] applied a signal packing technique which packs multiple periodic signals into a message; [28] proposed to reserve slots for aperiodic messages so that flexible medium access of the dynamic segment is preserved while QoS assurance can also be guaranteed. Both papers formulated NIP and decomposed the NIP problems into integer linear programming (ILP) problems. Another work [29] transformed the message scheduling problem for the static segment into a bin packing problem and again applied ILP to solve it. [32] proposed a heuristic to construct the communication schedule on the static segment of FlexRay systems. However, these papers only focused on message scheduling and neglected tasks in ECUs. The isolated message scheduling may severely limit the performance and feasibility of applications which consist of both tasks and messages. In contrast, our paper holistically investigates the scheduling of both tasks and messages on time-triggered systems.

The holistic scheduling on both tasks and messages has also been studied for time-triggered systems [20], [13], [18], [19], [11]. [20] applied constraint logic programming (CLP) to the scheduling and voltage scaling of low-power fault-tolerant hard real-time applications mapped on distributed heterogeneous embedded systems. [18] leveraged geometric programming (GP) to assign task and message periods for distributed automotive systems. [19] developed scheduling analysis approaches for hybrid event-triggered and time-triggered systems. [31] applied genetic algorithm techniques to solve the scheduling problem for FlexRay systems. The high complexity of solving mathematical programming limits the applicability and scalability of such methods.

The scheduling of task graphs in multiprocessors has been extensively studied in past decades. One widely applied type of algorithms is *list scheduling*. A list scheduling heuristic maintains a list of all tasks according to their priorities. It then iterates to pick tasks and schedule them onto selected processors. Some of the examples are the Highest Level First (HLF) [14], Earliest Task First (ETF) [15], Dynamic Critical Path (DCP) [16], and Mobility Directed (MD) [17] algorithms. As list scheduling approaches can provide high performance at a low cost, our paper presents algorithms based on list scheduling techniques. DCP is an efficient list scheduling algorithm for allocating task graphs on multiprocessors to minimize schedule length. One valuable feature of DCP is that the start times of the scheduled nodes are unfixed until all nodes have been scheduled. Our UST heuristic also applies this feature. The differences between DCP and UST are that DCP is not a real-time scheduling algorithm and cannot handle contention of different messages on communication resources, while UST is designed for real-time scheduling and can address the contention of different messages on time slots in time-triggered systems.

Another type of heuristic is clustering [26], [25], [24]. In this category, tasks are pre-clustered before allocation begins to reduce the size of the problem. Task clusters (instead of individual tasks) are then assigned to individual processors. [26] presented a clustering-based co-synthesis algorithm, which schedules periodic task graphs for hardware and software

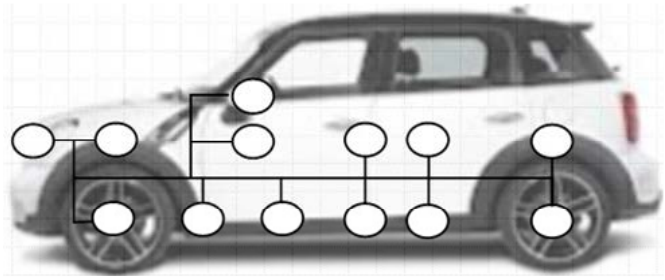


Fig. 1. A cluster of ECUs on a car

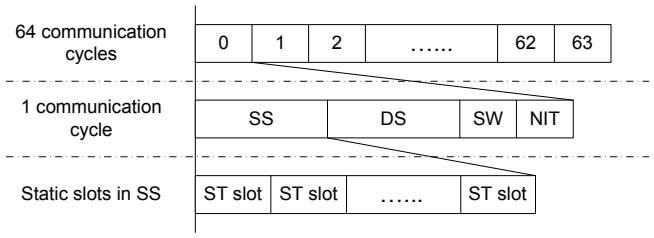


Fig. 2. FlexRay timing hierarchy.

co-synthesis on heterogeneous distributed embedded systems. [25] proposed a period-based approach to the problem of workload partitioning and assignment for large distributed real-time systems. [24] designed a static algorithm for allocating and scheduling components of periodic tasks across sites in distributed systems. In the problem discussed in this paper each task must be processed in a specific ECU; therefore, clustering may be useless since the tasks have been naturally clustered by their functionality. In this case, these clustering heuristics may lose their advantages when dealing with the problem discussed in this paper.

Optimal methods for some real-time task graph scheduling problems have also been proposed. [22] proposed an optimal B&B algorithm for allocating communicating periodic tasks to heterogeneous distributed real-time systems. [23] presented an optimal B&B algorithm for task assignment on multiprocessors subject to precedence and exclusion constraints. These are, however, applicable to only small task graphs.

III. Problem Formulation

A. System models

The target platform is a typical time-triggered in-vehicle system: a cluster of ECUs (i.e., hosts) that are connected via the FlexRay bus, as shown in Fig. 1. The operating system is non-preemptive. Each ECU can process particular tasks which exchange data via messages transferred on the bus. The operation of a FlexRay bus is based on repeatedly executed communication cycles with a fixed duration. A FlexRay cycle comprises a static segment (SS) and other segments such as a dynamic segment (DS). This paper focus on periodic applications, which only utilizes the static segment. Fig. 2 shows the timing hierarchy of 64 FlexRay cycles with an emphasis on the static segment. Each static segment consists of a fixed number of equal size static slots. Each static slot in each each cycle can only be uniquely assigned to one ECU to transfer one frame (i.e., message), but each static slot can be assigned to different ECUs in different cycles. The lengths of static slot T_s, static segment T_{ss}, and the communication cycle T_c are assumed to be known beforehand as previous papers have shown how to determine these parameters [10], [30].

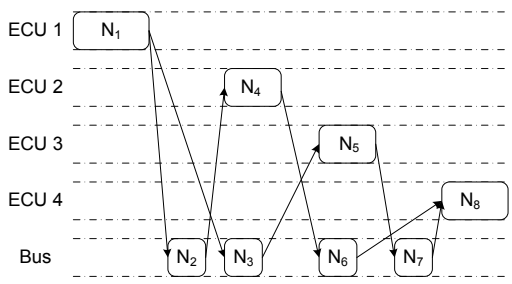


Fig. 3. An example of task graph running on ECUs.

B. Application Models

This paper assumes a periodic real-time task model where $G = \{g_1, g_2, ..., g_J\}$ is a set of J applications to be processed. Each application is independent from others. [1] Let $p(g_j)$ be the period of application $g_j \in G$ and L be the least common multiple of all $p(g_j)$s. The interval $[0, L)$ is called the hyper period. In one hyper period an application invokes $I(g_j) = \frac{L}{p(g_j)}$ times. Also, one hyper period spans multiple communication cycles. It suffices to analyze the behavior of the whole system only in one hyper period, since it will repeat for all hyper periods [22].

As shown in Fig. 3, an application can be modeled by a directed acyclic graph (DAG) comprising multiple nodes (i.e., vertexes), which are the smallest units to be scheduled, and edges, which specify precedence constraints. *Task graph*, *DAG* and *application* terms are interchangeably used in this paper. A node can be either a task (i.e., computation module) running on a particular ECU (e.g., a sensor, or an actuator) or a message (i.e., communication module) exchanged on the communication bus. Associated with each node n_i is its time cost $w(n_i)$ indicating the execution time on an ECU if the node is task, or the transmission time on the bus if the node is a message. In addition, since each ECU has its specific function, the host ECU of each task is known beforehand and processor selection appearing in conventional work is not needed. The host of node n_i is specified as $H(n_i)$. Further, messages nodes should be fit into static slots on the FlexRay bus. The transmission of a message cannot span two or more static slots.

An edge e_{ij} linking two nodes n_j and n_i specifies the precedence constraint between the nodes. That is, n_i should complete its execution before n_j starts. The edges incur no time cost. The source node n_i and the destination node n_j of the edge e_{ij} are called *parent* and *child* respectively. A node without parents is called an entry node while a node without

[1] In practice multiple applications may be related in the sense that they share some tasks or messages. In this case these related applications are merged/regarded as one application whose period is the least common multiple of these applications' periods.

children is called an exit node. Also, once n_i is scheduled onto $H(n_i)$, $prev(n_i)$ is the node scheduled immediately before n_i and $next(n_i)$ is the node scheduled immediately after n_i on $H(n_i)$. Moreover, a node n_i that must be finished before the start of another node n_j is called an *ancestor* of n_j, and n_j is called an *offspring* of n_p. Hence, the ancestors of n_i's parents and $prev(n_i)$ are also n_i's ancestors; the offspring of n_i's children and $next(n_i)$ are also n_i's offspring.

Each application has an offset $o(g_j)$, which indicates the start time of each invocation of g_j in one hyper period. That is, the start time of the k-th invocation of g_j is $o(g_j)+k*p(g_j)$ where $k = 0, 1, ..., I(g_j) - 1$. The offset of each application may vary between $[0, p(g_j))$. By default $o(g_j)$ is set as zero and it will be determined via the scheduling algorithm. Each node n_i may have a release time (i.e., input earliest start time, $EST^R(n_i)$). The k-th invocation of a node n_i (i.e., n_i^k) cannot start before $EST^R(n_i) + o(g_j) + k * p(g_j)$, where g_j is the application containing n_i. Also, each node n_i may have a deadline $d(n_i)$. The k-th invocation of a node n_i cannot finish after $d(n_i)+o(g_j)+k*p(g_j)$. The release time and deadlines denote the timing constraints imposed by schedule design and integration.

C. The Scheduling Problem

The main objective is to schedule all nodes in all invocations of all applications in one hyper period to guarantee that all invocations of all applications can satisfy their respective deadlines. Once this goal can be achieved, the scheduler may aim to minimize the length of the static segment which is actually used, denoted as T_{ss}^u, i.e., to minimize the number of used slots in the static segment. The unused bandwidth can either be used by the dynamic segment or be reserved for further schedule extension.

Parameter	Definition
$CPL_(g_i)$	the critical path length of application g_j
$d(g_j)$	relative deadline of application g_j
$d(n_i)$	relative deadline of node n_i
e_{ij}	edge linking two nodes n_i and n_j
$H(n_i)$	the host (ECU or bus) of node n_i
L	length of a hyper period
n_i	i-th node in a given application
n_i^k	k-th invocation of i-th node in a given application
$p(g_j)$	period of application g_j
$rank(g_j)$	the rank of graph g_j
T_c	length of a bus cycle
T_s	length of a slot in the static segment of the bus
T_{ss}	length of the static segment
$w(n_i)$	time cost of node n_i
Variable	**Definition**
$EST(n_i^k)$	the earliest start time of node n_i^k
$EST^G(n_i^k)$	EST value constrained by n_i^k's parents
$EST^P(n_i^k)$	EST value constrained by $prev(n_i^k)$
$EST^R(n_i)$	release time of node n_i
$EST^R(n_i^k)$	EST value constrained by input requirements
$EST(n_i^k, m)$	EST value of n_i^k after it is inserted into m-th position
$LST(n_i^k)$	the latest start time of node n_i^k
$LST^G(n_i^k)$	LST value constrained by n_i^k's children
$LST^P(n_i^k)$	LST value constrained by $next(n_i^k)$
$LST^R(n_i^k)$	LST value constrained by input requirements
$LST(n_i^k, m)$	LST value of n_i^k after it is inserted into m-th position
$next(n_i^k)$	next node of n_i
$o(g_j)$	the offset of application g_j
$RM(n_i^k)$	relative mobility of n_i
$prev(n_i^k)$	previous node of n_i
$priority(g_j)$	priority of application g_j
$ST(n_i^k)$	start time of n_i^k
T_{ss}^u	length of the static segment which is actually used

TABLE I
NOTATIONS AND TERMINOLOGY

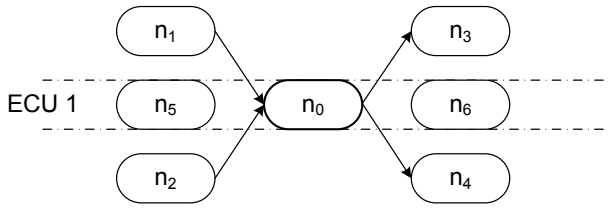
IV. The Proposed Algorithms

This section describes the proposed algorithm whose pseudo code is shown in Algorithm 1. The algorithm works by iteratively selecting applications and scheduling individual nodes in the selected applications via UST, while ROM and BPP serve as complements to enhance the schedulability of the applications. The major components and their features of the algorithm are summarized as follows:

- Section IV-A introduces two attributes: *earliest start time* (EST) and *latest start time* (LST), which are assigned to each node and will be used by UST described below.
- Section IV-B describes the *application selection* procedure, which orders and selects applications for scheduling.
- Section IV-C details the UST scheduling heuristic, which flexibly schedules nodes of each selected application. When the nodes are being scheduled (i.e., ordered), the start times of the scheduled nodes are not fixed. The unfixed scheduling policy offers more opportunities to insert nodes into proper positions between scheduled nodes.
- Section IV-D presents ROM. Upon a confliction in time allocation, ROM reschedules the conflicted application with an adjusted offset such that the scheduling of different applications can be staggered to avoid conflictions.
- Section IV-E describes BPP. Once ROM cannot help to eliminate conflictions, BPP promotes the priority of the conflicted application and backtracks previously scheduled applications to create space for the conflicted application.
- Section IV-F presents a *bandwidth optimization* procedure. Once all nodes are successfully ordered, the scheduler determines a final schedule that can optimize the bandwidth utilization of the bus while the deadline requirements of all scheduled nodes are still satisfied.

A. EST and LST

Since the nodes are not allocated fixed start times, two attributes are introduced for each node: *earliest start time* (EST) and *latest start time* (LST), which are the *lower bound* and *upper bound* on the start time of a node. The EST and LST of a node can reflect its mobility since its actual start time can slide between its EST and LST. Each invocation of a node is separately scheduled. Hence, each node n_i^k (i.e., k-th invocation of node n_i) has its own EST and LST. In the following, the EST and LST of node n_i^k belonging to g_j are derived.



node	$\mathbf{n_0}$	n_1	n_2	n_3	n_4	n_5	n_6
size	10	5	2	NA	NA	7	NA
EST	?	8	10	NA	NA	2	NA
LST	?	NA	NA	50	40	NA	30

Fig. 4. An example for calculating EST and LST of node n_0.

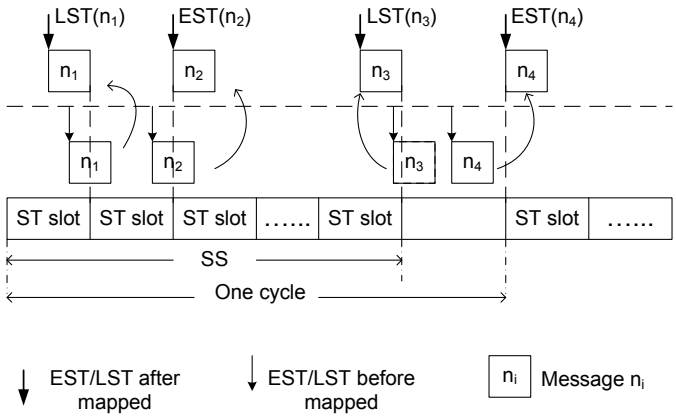


Fig. 5. Examples for calculating EST and LST for messages.

The EST of a node n_i^k is defined as:

$$EST(n_i^k) = \max\{EST^R(n_i^k), EST^G(n_i^k), EST^P(n_i^k)\} \quad (1)$$

where $EST^R(n_i^k)$ is the EST value constrained by input requirements; $EST^G(n_i^k)$ is the EST value constrained by n_i^k's parent nodes; $EST^P(n_i^k)$ is the EST value constrained by $prev(n_i^k)$. By default $o(g_j)$ is set as zero and it can be modified in Section IV-D. One can write $EST^R(n_i^k)$ as follows:

$$EST^R(n_i^k) = EST^R(n_i) + o(g_j) + k * p(g_j) \quad (2)$$

If $EST^R(n_i)$ is not specified, $EST^R(n_i) = 0$. Also, one can write $EST^G(n_i^k)$ as follows:

$$EST^G(n_i^k) = \max_{1 \le p \le P} \{EST(n_{i_p^k}) + w(n_{i_p^k})\} \quad (3)$$

where n_i^k has P parents and $n_{i_p^k}$ is the p-th parent node. If n_i^k is an entry node, then $EST^G(n_i^k) = o(g_j) + k * p(g_j)$. Equation (3) states that the EST of a node n_i^k is no greater than the earliest ready time of all its parents. $EST^P(n_i^k)$ is the earliest finish time of the node that is scheduled immediately before n_i^k on the same host and can be written as:

$$EST^P(n_i^k) = EST(prev(n_i^k)) + w(prev(n_i^k)) \quad (4)$$

If n_i^k is scheduled as the first node on the processor, $EST^P(n_i^k) = 0$. In addition, if n_i^k has not been scheduled onto its host, $EST^P(n_i^k) = 0$.

Fig. 4 shows an example for calculating EST and LST of a node n_0 where the superscript k is omitted and $EST^R(n_0)$ is given as zero. In this example, only a small part of a whole application, i.e., only directly related nodes to n_0, are depicted. Nodes n_1 and n_2 are n_0's parents and nodes n_3 and n_4 are its children; n_5 and n_6 are scheduled immediately before and after n_0, respectively on n_0's hosted ECU. The table in Fig. 4 provides all required information for calculating n_0's EST and LST, i.e., the sizes, EST, and LST of the above nodes. According to Equations (3) and (4), $EST^G(n_0) = \max\{EST(n_1) + w(n_1), EST(n_2) + w(n_2)\} = \max\{5+8, 2+10\} = 13$; $EST^P(n_0) = EST(n_5) + w(n_5) = 7 + 2 = 9$. Consequently, $EST(n_0) = \max\{EST^R(n_0), EST^G(n_0), EST^P(n_0)\} = \max\{0, 13, 9\} = 13$.

If n_i^k is a message node, $EST(n_i^k)$ is tailored so that the message can be fit into a slot in the communication bus. Let $ST^{slot}(t)$ be the start time of the slot in which time t resides.

$$ST^{slot}(t) = \lfloor \frac{t}{T_c} \rfloor T_c + \lfloor \frac{t \mod T_c}{T_s} \rfloor T_s \quad (5)$$

Once $EST(n_i^k)$ is computed in Equation (1), it can be tailored for a message node as:

$$EST(n_i^k) = \begin{cases} \lceil \frac{EST(n_i^k)}{T_c} \rceil T_c & case\,1 \\ ST^{slot}(EST(n_i^k)) + T_s, & otherwise \end{cases} \quad (6)$$

where case 1 is that $EST(n_i^k)$ computed in Equation (1) falls outside the range of static segments ($EST(n_i^k) \mod T_c > T_{ss}$ and $EST(n_i^k) + w(n_i) \mod T_c > T_{ss}$). Thus $EST(n_i^k)$ is mapped to the start time of the first static slot of the next communication cycle. Fig. 5 shows examples for mapping the EST of message nodes into static slots, where EST^T denotes the EST value computed in Equation (1) and EST^M denotes the mapped EST value in Equation (6). In Fig. 5, the mapping of message n_4 shows this case. Otherwise, $EST(n_i^k)$ is mapped to the start time of the next static slot, as shown by message n_2 in Fig. 5.

$EST(n_i^k)$ can be computed once the EST of all n_i^k's parents and $prev(n_i^k)$ are known. Hence, the EST of all nodes can be calculated by traversing the DAGs in a top-down manner beginning from the entry nodes that are not scheduled or are scheduled as the first nodes on their respective hosts. Consequently, when all the EST of n_i^k's parents and $prev(n_i^k)$ are available, the EST of n_i^k can be computed.

The LST of a task node n_i^k is defined as follows:

$$LST(n_i^k) = \min\{LST^R(n_i^k), LST^G(n_i^k), LST^P(n_i^k)\} \quad (7)$$

where $LST^R(n_i^k)$ is the LST value constrained by input requirements; $LST^G(n_i^k)$ is the LST value constrained by n_i^k's children; $LST^P(n_i^k)$ is the LST value constrained by $next(n_i^k)$. One can write $LST^R(n_i^k)$ as follows:

$$LST^R(n_i^k) = o(g_j) + k * p(g_j) + d(n_i) - w(n_i) \quad (8)$$

If $d(n_i)$ is not specified, $LST^R(n_i^k) = \infty$. Also, one can write $LST^G(n_i^k)$ as follows:

$$LST^G(n_i^k) = \min_{1 \le q \le Q} \{LST(n_{i_q^k}) - w(n_i)\} \quad (9)$$

where n_i^k has Q children and $n_{i_q^k}$ is the q-th child. If n_i^k is an exit node, $LST^T(n_i^k) = \infty$.

$LST^P(n_i^k)$ is constrained by the start time of $next(n_i^k)$. Accordingly, one can write $EST^G(n_i^k)$ as:

$$LST^P(n_i^k) = LST(next(n_i^k)) - w(n_i) \quad (10)$$

If n_i^k has not been scheduled or n_i^k is scheduled as the last node on its host, $LST^P(n_i^k) = \infty$.

In the example of Fig.4, $LST^R(n_0)$ is given as ∞; $LST^G(n_0) = \min\{LST(n_3) - w(n_0), LST(n_4) - w(n_0)\} = \min\{50 - 10, 40 - 10\} = 30$, $LST^P(n_0) = LST(n_6) - w(n_0) = 30 - 10 = 20$. As a result, $LST(n_0) = \min\{LST^R(n_0), LST^G(n_0), LST^P(n_0)\} = \min\{\infty, 30, 20\} = 20$.

If n_i^k is a message node, one can tailor $LST(n_i^k)$ for a message node after it is computed in Equation (7) as:

$$LST(n_i^k) = \begin{cases} \lfloor \frac{LST(n_i^k)}{T_c} \rfloor T_c + T_{ss} - w(n_i), & case\,1 \\ ST^{slot}(LST(n_i^k)) + T_s - w(n_i), & otherwise \end{cases} \quad (11)$$

where case 1 is that $LST(n_i^k)$ computed in Equation (7) falls outside the range of static segments. Thus $LST(n_i^k)$ is mapped to the last static slot of the last communication cycle. Fig. 5 shows examples for mapping the LST of message nodes into static slots, where LST^T denotes the LST value computed in Equation (7) and LST^M denotes the mapped LST value in Equation (11). In Fig. 5, the mapping of message n_3 shows this case. Otherwise, $LST(n_i^k)$ is mapped to the last static slot, as shown by message n_1 in Fig. 5. Similar to the computation of $EST(n_i^k)$, $LST(n_i^k)$ can also be computed by traversing the task graphs in a bottom-up manner.

B. Application Selection

The algorithm iterates to schedule all nodes of all invocations of each application. Since it is favoured to first schedule hard applications, the first step is to obtain the priorities of the applications and order them according to their priorities. The priority of an application g_j (denoted as $priority(g_j)$) is initially set as the rank of g_j, which is defined as:

$$rank(g_j) = \frac{p(g_j) + d(g_j)}{CPL(g_j)} \quad (12)$$

where $CPL(g_j)$ is the critical path length of the task graph and is defined as:

$$CPL(g_j) = \max_i \{EST(n_i^0) + w(n_i)\} \quad (13)$$

In addition, $d(g_j)$ is the relative deadline of g_j which represents its overall urgency (i.e., hardness of timing constraints) and $d(g_j)$ is defined as:

$$d(g_j) = \min_{n_i \in g_j} \{d(n_i) - LST(n_i^0)\} \quad (14)$$

Hence, $rank(g_j)$ can be obtained after $EST(n_i^0)$ and $LST(n_i^0)$ are calculated via Equations (1) and (7) for all nodes. The intuition to set $rank(g_j)$ as the initial priority is that a longer critical path length probably means that the graph is harder to schedule within limited space while a longer period and a longer deadline usually implies larger space for the nodes in the graph to be flexibly scheduled. It may be noticed that the priority can be modified as described in Section IV-E. The task graphs are sorted by the ascending order of their priorities. Accordingly, a task graph with the smallest priority is first selected for scheduling (Line 7 of Algorithm 1).

C. Node Scheduling (The UST Algorithm)

The UST algorithm flexibly schedules all nodes of all invocations of a selected application g_j (Lines 8-14 of Algorithm 1). When the nodes are being scheduled, the start times of the scheduled nodes are not fixed. Hence, the nodes are actually "clustered" together in a linear order. The unfixed scheduling policy offers more opportunities to insert nodes into proper positions between scheduled nodes. The flexibility offered by the algorithm is a significant advantage when compared to previous list scheduling algorithms (e.g., [26], [24], [14]) that assign fixed start times in the process of scheduling.

Since the start time of the scheduled nodes are not fixed when they are being scheduled, the scheduled nodes are actually ordered on their hosts. The only constraint is that the total order among the scheduled nodes will not be affected by the subsequent scheduling. While preserving the linear order of the scheduled nodes, the EST and LST values of the nodes can be updated in each round of node scheduling (i.e., ordering).

The algorithm iterates two steps, a *node selection* step for dynamically selecting nodes, and a *node insertion* step for scheduling selected nodes onto their hosts. The node with the smallest *relative mobility* value in g_j is selected for scheduling (Line 11 of Algorithm 1). The relative mobility of a node $RM(n_i^k)$ is defined as:

$$RM(n_i^k) = \frac{LST(n_i^k) - EST(n_i^k)}{w(n_i)} \quad (15)$$

Once a node n_i^k is selected, it will be scheduled on its host $H(n_i)$. Suppose a set of M nodes $\{n_0, ..., n_{M-1}\}$ have been scheduled on $H(n_i)$. Hence n_i^k may be scheduled onto $M+1$ positions, i.e., position $m \in [0, M]$ between two consecutively scheduled nodes n_{m-1} and n_m. Virtual nodes n_{-1} and n_M are used for the convenience of denoting the first and the last slots. For virtual nodes n_{-1} and n_M, let $EST(n_{-1}) = 0$, $w(n_{-1}) = 0$, $LST(n_M) = L$, and $w(n_M) = 0$. In order not to violate the precedence constraints among nodes, n_i^k must not be scheduled before its ancestors, or after its offsprings. If a position satisfies this constraint, it is called a candidate position. Let $EST(n_i^k, m)$ and $LST(n_i^k, m)$ denote the respective EST and LST values of n_i^k if it is scheduled onto m-th position on $H(n_i)$ (i.e., between nodes n_{m-1} and n_m). One can write $EST(n_i^k, m)$ as:

$$EST(n_i^k, m) = \max\{EST^G(n_i^k), EST(n_{m-1}) + w(n_{m-1})\} \quad (16)$$

Also, one can write $LST(n_i^k, m)$ as:

$$LST(n_i^k, m) = \min\{LST^G(n_i^k), LST(n_m) - w(n_i)\} \quad (17)$$

Theorem 1: If the current partial schedule of a set of nodes S is feasible, after a new node n_i^k is scheduled onto a candidate position m on $H(n_i)$, the new partial schedule for $S' \leftarrow S \cup \{n_i^k\}$ is still feasible provided:

$$LST(n_i^k, m) - EST(n_i^k, m) \geq 0 \quad (18)$$

Proof: For the convenience of proof, insert n_i^k on position m and accordingly update n_i^k's ancestors and offspring. In S, any node n_j's start time can be feasibly set as n_j's EST if all n_j's ancestors start time is set as their EST. Similarly, in S any node n_j's start time can be feasibly set as n_j's LST if all n_j's offspring's start time is set as their LST. In this case, after recursively setting all n_i^k's ancestors' start time as their EST and setting all n_i^k's offspring's start time as their LST, the schedulability of all nodes in S are kept. Afterwards, n_i^k can be scheduled on position m between the range $[EST(n_i^k, m), LST(n_i^k, m)]$ if $LST(n_i^k, m) > EST(n_i^k, m)$. ■

Among all candidate positions satisfying Equation (18), n_i^k is inserted into the one that can maximize $LST(n_i^k, m) - EST(n_i^k, m)$ (Lines 12-14 of Algorithm 1). That is, n_{m-1} becomes the previous node of n_i^k ($prev(n_i^k)$) if it exists, and n_m becomes the next node of n_i^k ($next(n_i^k)$) if it exists. Such a position can help to preserve the flexibility of n_i^k and may ease the scheduling of latter nodes.

After scheduling the node, the algorithm updates EST and LST for all nodes and continues selecting and scheduling nodes. By iterating these steps (Lines 9-14 of Algorithm 1), all nodes in g_j can be ordered accordingly. Afterwards, another applications is selected for scheduling. Once all applications are successfully scheduled, a final schedule is produced by simply setting the start time of each node, defined as $ST(n_i^k)$, as $ST(n_i^k) \leftarrow EST(n_i^k)$ (Line 35 of Algorithm 1).

D. Rescheduling with Offset Modification (ROM)

Due to the complex timing constraints in system integration, conflicts may frequently occur in node assignment, especially when the offset (release time) of each application is limited to be zero. It may be noticed that in many prior studies (e.g., [22], [23], [26]), offsets were simply fixed as zero, which might incur poor schedulability. If multiple applications can start at different offsets such that the time allocations of different applications can be staggered, the schedulability can be improved even under complex timing constraints. To this end, this paper designs the ROM approach, which reschedules conflicted applications with adjusted offsets to avoid the conflicts (Lines 24-29 of Algorithm 1).

Recall that the offset of each application is initially set as zero. Once no feasible position can be found for scheduling n_i^k via UST, conflictions in time allocation have occurred. To eliminate conflictions, ROM is applied to reschedule g_j to a new offset $o(g_j)$ such that n_i^k can be inserted into a slack (i.e., capable) slot on $H(n_i)$. The slackness of m-th slot is defined as:

$$\delta(m) = LST(n_m) - w(n_i) - EST(n_{m-1}) - w(n_{m-1}) \quad (19)$$

Actually $\delta(m)$ is the difference between the EST specified by n_{m-1} and LST specified by n_m for n_i^k. If $\delta(m) < 0$, the slot cannot hold n_i^k. But if any candidate position m exists such that $\delta(m) \geq 0$, the algorithm will modify $o(g_j)$ as:

$$o(g_j) = o(g_j) + (EST(n_{m-1}) + w(n_{m-1}) + LST(n_m) - w(n_i) - EST^G(n_i^k) - LST^G(n_i^k))/2. \quad (20)$$

where m is the position that $\delta(m)$ is the maximum among all candidate positions. Such an update enables that the mobility

Algorithm 1 The Scheduling Algorithm

1: Input: a set of J applications
2: Output: a schedule for the applications, i.e., start times ($ST(n_i^k)$) are determined for all nodes of the applications
3: $\forall g_j$, compute $rank(g_j)$ and initialize $priority(g_j)$
4: sort all applications by ascending order of $priority(g_j)$
5: outerLoop:
6: **while** not all applications are scheduled **do**
7: pick next application g_j from the application list for scheduling
8: **while** not all nodes of all invocations of g_j are scheduled **do**
9: update EST and LST for all nodes via Equations (1), (6), (7), and (11)
10: update ancestors and offspring for all nodes
11: select a node n_i^k with the smallest $RM(n_i^k)$
12: find a position m that maximizes $LST(n_i^k, m) - EST(n_i^k, m)$ among all candidate positions
13: **if** the position is found **then**
14: schedule n_i^k in the position
15: **else if** $cnt_{res} > max_{res}$ or a feasible δ_m is not found **then**
16: $cnt_{res} \leftarrow 0$
17: **if** exit conditions are fulfilled **then**
18: exit without a feasible solution
19: **end if**
20: $cnt_{back} \leftarrow cnt_{back} + 1$
21: execute a long-distance backtrack
22: update $priority(g_j)$ via Equation(21)
23: continue outerLoop:
24: **else**
25: $cnt_{res} \leftarrow cnt_{res} + 1$
26: update $o(g_j)$ via Equation(20)
27: backtrack all nodes of all invocations of g_j
28: continue outerLoop:
29: **end if**
30: **end while**
31: remove g_j from the application list
32: **end while**
33: update EST and LST for all nodes
34: if required, call Algorithm 2 to optimize bandwidth
35: $\forall n_i^k$, $ST(n_i^k) \leftarrow EST(n_i^k)$

of n_i^k given by its parents and children (i.e., $LST^G(n_i^k) - EST^G(n_i^k)$) probably fits m-th slot in the following runs.

Then, all scheduled nodes in g_j are backtracked, i.e., they are now unscheduled. Afterwards, the algorithm continues the node scheduling procedure to reschedule g_j.

E. Long-Distance Backtrack and Priority Promotion (BPP)

Once ROM is not helpful, BPP backtracks previously scheduled task graphs to create space for the failed one (Lines 15-23 of Algorithm 1). The backtracking technique is widely used in exhaustive methods (e.g., branch and bound (B&B)) [22], [23] which are, however, applicable to only small task graphs. Accordingly, some prior work [27] limited the number/level of backtracks to refrain time consumption. Since only one node was backtracked in each step, the limited local exhaustive search in such papers is probably ineffective for large task graphs.

In contrast, to make a trade-off between time and performance, BPP may backtrack multiple applications (rather than one node) at one time depending on previous progresses. That is, if the total number of offset modification and rescheduling for the failed application g_j (denoted as cnt_{res}) reaches a predefined limit (denoted as max_{res}) or a feasible δ_m cannot be found, the algorithm executes a long-distance backtrack, which not only backtracks g_j, but also backtracks multiple previously scheduled applications to create space for g_j. The number of applications backtracked, denoted as $back_{limit}$, is initially set as 1. If g_j has ever failed previously, $back_{limit}$ is doubled; otherwise it is reset. For each backtracked application, all nodes are backtracked. The algorithm repeatedly backtrack scheduled applications from the tail of the scheduled list until a number of $back_{limit}$ applications have been backtracked or the scheduled list is empty.

In addition, according to Equation (12), $priority(g_j) \leq 2$ and an application with $priority(g_j) = 2$ has the highest priority. Hence the priority of g_j is updated as:

$$priority(g_j) = priority(g_j)/2 + 1 \tag{21}$$

This significantly boosts the priority of g_j, which is probably hard to schedule, and places it into a front position for scheduling in the next run. Afterwards, the algorithm continues to select applications and scheduling nodes via UST. The algorithm terminates once a feasible solution is found or the following conditions are fulfilled. All recently N_F failed applications have ever failed before or the total number of long-distance backtracks cnt_{back} reaches max_{back}. In our simulations, N_F is set as 5 and max_{back} is set as 20.

F. Schedule Determination with Bandwidth Optimization

This subsection provides an approach to derive a schedule with bandwidth optimization which can be used once such optimization is needed (Line 34 of Algorithm 1). Algorithm 2 depicts the pseudo code of the bandwidth optimization method. After all nodes are successfully ordered in Algorithm 1, Algorithm 2 determines a feasible schedule that can minimize T_{ss}^u. Since T_{ss}^u is a multiple of T_s, there are only limited choices for T_{ss}^u as $T_{ss} < T_c$. In this case, a binary search approach is applied to search for the minimum T_{ss}^u that is feasible for a schedule. Let T_{ss}^{low} denote the currently largest infeasible value of T_{ss}^u and let T_{ss}^{high} denote the currently smallest feasible value of T_{ss}^u. Since a successful schedule exists when $T_{ss} = T_{ss}^{max}$, initially let $T_{ss}^{high} = T_{ss}^{max}$ and $T_{ss}^{low} \leftarrow T_s$. The while loop in Algorithm 2 iteratively updates T_{ss}^{low} and T_{ss}^{high} and searches for a feasible T_{ss}^u among the range (T_{ss}^{low}, T_{ss}^{high}) by updating EST and LST for each checked T_{ss}^u. If a candidate T_{ss}^u is feasible, after EST and LST are updated for all nodes, $EST(n_i) \leq LST(n_i)$ holds for each node n_i. Finally, the actual start time of each node is set as its EST updated after setting $T_{ss}^u \leftarrow T_{ss}^{high}$ (Lines 16 and 17 of Algorithm 2).

Algorithm 2 Schedule Determination with Bandwidth Optimization

1: $T_{ss}^{high} \leftarrow T_{ss}^{max}$
2: $T_{ss}^{low} \leftarrow T_s$
3: **while** true **do**
4: **if** $T_{ss}^{low} + T_s < T_{ss}^{high}$ **then**
5: $T_{ss}^u \leftarrow T_s \lfloor \frac{T_{ss}^{low} + T_{ss}^{high}}{2T_s} \rfloor$
6: update EST and LST for all nodes
7: **if** $\forall n_i$, $EST(n_i) \leq LST(n_i)$ **then**
8: $T_{ss}^{high} \leftarrow T_{ss}^u$
9: **else**
10: $T_{ss}^{low} \leftarrow T_{ss}^u$
11: **end if**
12: **else**
13: break
14: **end if**
15: **end while**
16: $T_{ss}^u \leftarrow T_{ss}^{high}$
17: update EST and LST for all nodes

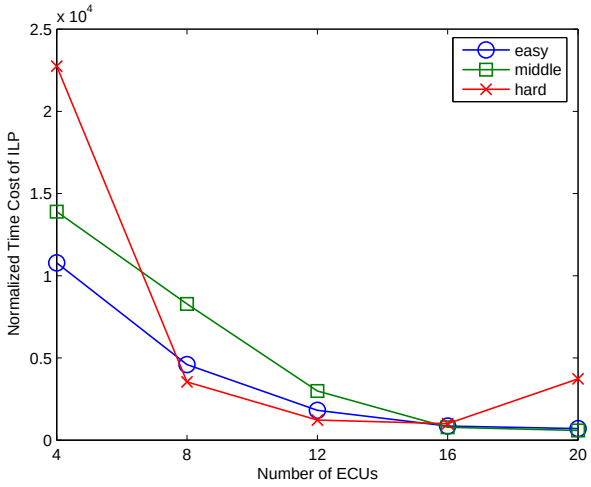
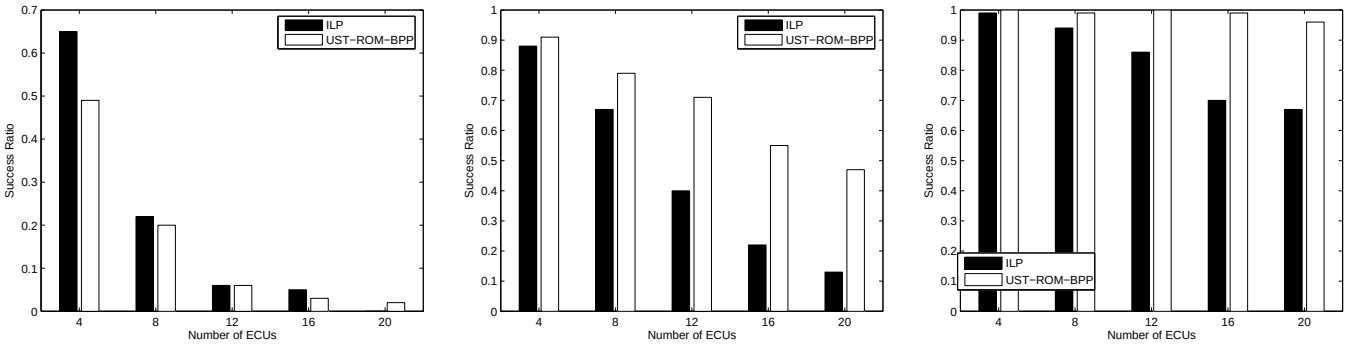


Fig. 7. Normalized time cost of ILP.

V. Performance Evaluation

In order to assess the effectiveness of the proposed scheduling algorithm, this section presents a performance evaluation study for scheduling a number of real-time applications (i.e.,



(a) Easy experiment (b) Middle experiment (c) Hard experiment

Fig. 6. Success ratio for the comparison of ILP and UST-ROM-BPP.

task graphs). The success ratio is used as the major performance metric. It is defined as the the number that an algorithm successfully schedules all application to the number of total experiments.

The system configurations are set as follows: A set of synthetic applications are randomly generated based on realistic cases. Basic parameters including the periods, deadlines, and topology of the applications and the sizes of tasks and messages are randomized according to sampled values from industrial cases. Specifically, the period of each application is varied among [5ms, 10ms, 20ms, 40ms]. The length of one hyper schedule period is thus 40ms. The duration of a cycle of the communication bus is 5ms and the duration of the available static segment per cycle is 3.75ms. The duration of a communication slot is set as 0.0625ms. The average cost of a task is 2ms. Experiments with three different difficulties, easy, middle and hard, are conducted. For each difficulty level, one figure with five data points is plotted to exhibit the performance of a set of algorithms on different system scales to evaluate the scalability of the algorithms. In the following figures of results, the horizontal axis marks the number of ECUs, which is varied among the range [4, 8, 12, 16, 20]. That is, the number of ECUs on the horizontal axis denotes the scale of the experiments as the number of ECUs and the number of nodes simultaneously grows in each experiment.

Since the proposed algorithms comprise three major parts (i.e., UST, ROM, and BPP), to understand the merits of our algorithms, the results of three algorithm combinations are separately presented. The first algorithm, denoted as UST, only enables UST and disables both ROM and BPP. The second algorithm, denoted as UST-ROM, enables UST and ROM and disables BPP. The third algorithm, denoted UST-ROM-BPP, enables all three proposed approaches.

As a first case, UST-ROM-BPP is compared with an ILP solution formulated in [13]. In the ILP formulation an eCos-based operating system and the FlexRay 3.0 model are applied. The ILP formulation is then solved by the CPLEX ILP solver [33]. Since ILP is very time-consuming and thousands of input cases are simulated, a time-out of one hour is set for ILP to guarantee that the simulations can stop within acceptable time. As the results below show, the proposed algorithm can deliver more competitive performance within few seconds, which renders the one-hour timeout for ILP sufficiently long for performance comparison. In this case, as the scale of experiments increases, ILP may not deliver the optimal solution within the time-out. The results on three difficulty levels are shown in Fig. 6. Among the three experiments, the number of nodes per ECU is the largest and deadlines are the most urgent in the hard experiment, while in the easy experiment the number of nodes per ECU is the smallest and deadlines are the least urgent. The average ratio of deadline to period of each application is 0.82, 0.71, and 0.6, for easy, middle, and hard cases, respectively.

From Fig. 6 one can observe that UST-ROM-BPP outperforms ILP in the easy and middle experiments. Specifically, as the number of ECUs increases, the performance gap between UST-ROM-BPP and ILP significantly enlarges, which is due to the fact that ILP is unscalable. As the number of ECUs increases, it becomes more difficulty for ILP to search for solutions within the timeout. Such gaps do not exist in Fig. 6(c), showing that both UST-ROM-BPP and ILP are hindered in hard cases. Also, Fig. 7 shows the time cost of ILP normalized to the time cost of UST-ROM-BPP in the three experiments. As Fig. 7 shows, UST-ROM-BPP consumes much less time than ILP, i.e., the time consumed by ILP is about 1000 - 10,000 times of the time consumed by UST-ROM-BPP. Since the performance of UST-ROM-BPP is at least as good as ILP in Fig. 6, the advantages of UST-ROM-BPP over ILP in speed and efficiency are quite obvious.

As ILP is unscalable, in the following experiments the proposed algorithms are compared with 3 peer list scheduling heuristics, ETF [15], HLF [14], MD [17], which are adapted from previous algorithms. Because HLF outperforms ETF and MD in our experiments, to further evaluate the effectiveness of the proposed algorithms, by replacing UST with HLF in UST-ROM and UST-ROM-BPP, another two algorithm combinations, HLF-ROM and HLF-ROM-BPP, are generated. In HLF-ROM, HLF is used for node scheduling and ROM is tuned to adapt HLF. In HLF-ROM-BPP, both ROM and BPP are enabled. By comparing HLF, HLF-ROM, HLF-ROM-BPP, UST, UST-ROM, and UST-ROM-BPP, one can evaluate the benefits of UST, ROM, and BPP.

Then, experiments with three different difficulties, easy, middle, and hard, are conducted and the results are shown

in Fig. 8. The average ratio of deadline to period of each application is 0.82, 0.77, and 0.7, for easy, middle, and hard cases, respectively. It may be noticed that the inputs for these experiment are easier than those of the first experiments for comparing UST-ROM-BPP and ILP because the performance of the peer list scheduling algorithms is much poorer than UST-ROM-BPP and ILP.

The following observations are made from Fig. 8. Firstly, UST-ROM-BPP significantly delivers the best performance among all evaluated algorithms. Specifically, for the middle experiment (Fig. 8(b)), the performance gap between UST-ROM-BPP and others is huge. That is, the success ratio of UST-ROM-BPP is kept above 0.7 while for other algorithms the success ratio is no larger than 0.25. Similarly, for the hard experiment (Fig. 8(c)), the success ratio of UST-ROM-BPP is kept above 0.3 while for other algorithms the success ratio is no larger than 0.1. These demonstrate the effectiveness of the proposed three approaches. Secondly, UST constantly outperforms HLF, ETF, and MD on success ratio by a clear margin. This demonstrates that the unfixed start time of UST offers more opportunities to flexibly insert nodes into proper positions between scheduled nodes and thus greatly enhances overall schedulability. Thirdly, UST-ROM and UST-ROM-BPP outperform their counterparts, HLF-ROM and HLF-ROM-BPP, respectively. These again support the above conclusion that UST is advantageous in node scheduling. Fourthly, UST-ROM and HLF-ROM outperform UST and HLF, respectively. This demonstrates the effectiveness of ROM in enhancing schedulability. Fifthly, UST-ROM-BPP and HLF-ROM-BPP outperform UST-ROM and HLF-ROM, respectively. This demonstrates the effectiveness of BPP in enhancing schedulability. Finally, the above observations are consistent among three different difficulties. These further demonstrate the effectiveness of the proposed approaches.

The time cost to achieve high performance is not free. Fig. 9 shows the corresponding time cost of the algorithms for the easy experiment (i.e., Fig. 8(a)). It is shown that UST, UST-ROM and UST-ROM-BPP require much more time than other algorithms. UST-ROM-BPP, which outperform others in success ratio, consumes the most time among all evaluated heuristic algorithms. Nevertheless, UST-ROM-BPP is still quite scalable when its time cost is compared to that of ILP, as shown in Fig. 7. Due to limited space, only the time cost of the easy experiment is shown in the paper and the time cost of other experiments are quite similar to this one.

Fig. 10 depicts the bandwidth saved (i.e., $1 - \frac{T^u_{ss}}{T_{ss}}$) by UST-ROM-BPP in the above three experiments (i.e., Fig. 8). It is shown that the algorithm saves up to 17% of bandwidth in the experiments. Specifically, the algorithm saves less bandwidth in harder cases. This may be due to the fact that timing constraints are more urgent in harder cases, which limits the room for bandwidth optimization. In addition, as the number of ECUs increases, the bandwidth that can be saved decreases. A plausible explanation is that when the number of ECUs grows, there are more time constraints, which may limit the space for bandwidth optimization.

In the above experiments, each application has only one deadline. To evaluate the performance of the algorithms under

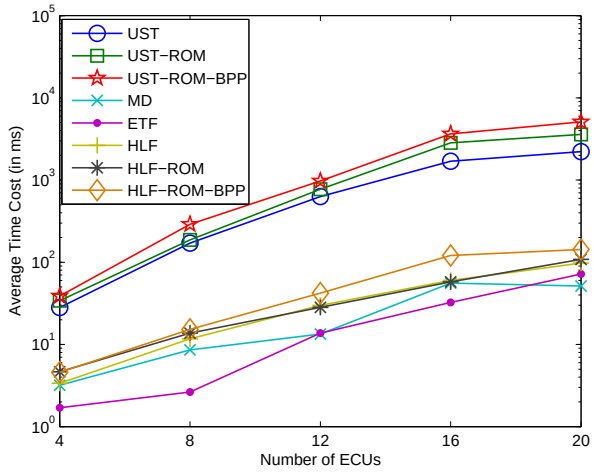


Fig. 9. Average time cost of the heuristics.

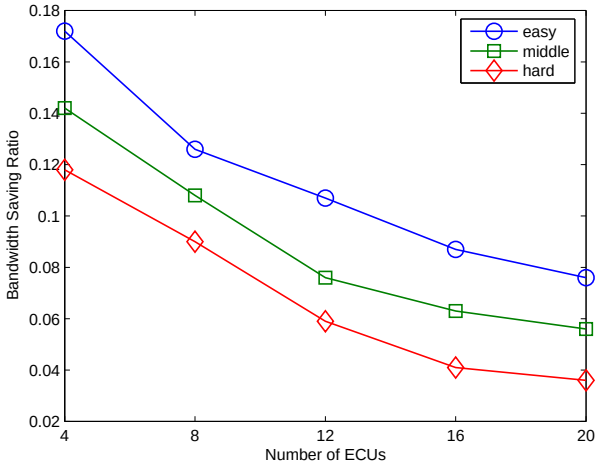


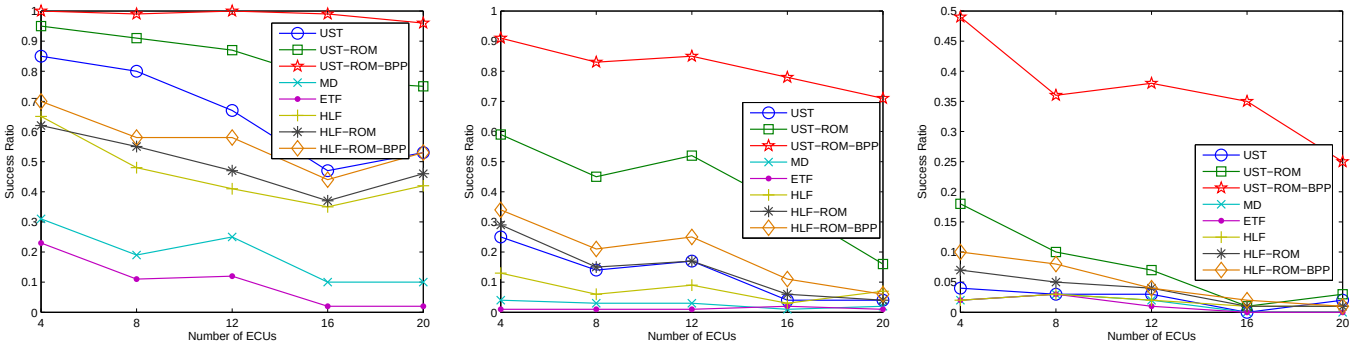
Fig. 10. Bandwidth saving ratio of UST-ROM-BPP.

complex timing constraints, in the following two experiments, the input cases of the above easy experiment are reused while complex timing constraints are added. Fig. 11 shows the corresponding success ratio of the two experiments. In the first one (Fig. 11(a)), 50% of entry and exit nodes and 20% of normal task nodes have constrained release time and deadlines. In the second experiment (Fig. 11(b)), 40% of entry and exit nodes have constrained release time and deadlines and another 15% of entry and exit nodes have given start times, i.e., the release time is equal to the deadline for such nodes.

The results in Fig. 11 basically support the observations made from Fig. 8. In addition, by comparing Fig. 11 with Fig. 8(a), one can observe that the degradation of success ratio of UST-ROM-BPP is smaller than that of UST and UST-ROM. This demonstrates that ROM and BPP can effectively tolerate conflictions brought by complex timing requirements.

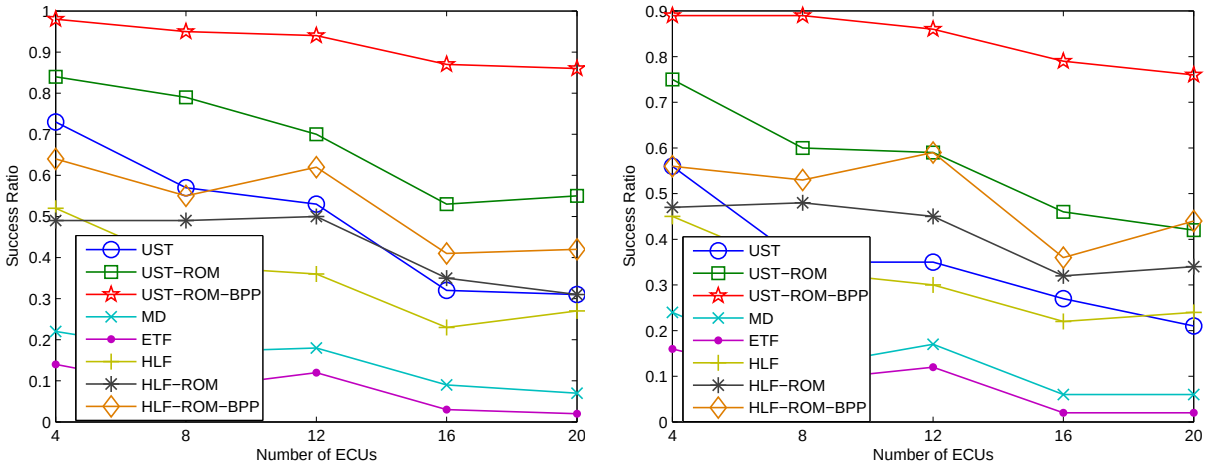
VI. Conclusions

This paper has studied an important scheduling problem for holistically handling both tasks and messages in time-triggered automotive systems. This paper has formulated novel



(a) Easy experiment (b) Middle experiment (c) Hard experiment

Fig. 8. Results for the comparison of the heuristics.



(a) Type 1 (b) Type 2

Fig. 11. Success ratio of experiments with complex timing constraints.

models for practical system design and integration in automotive industries. This paper has presented the UST algorithm that schedules tasks and messages in a flexible way to enhance schedulability. In addition, to tolerate assignment conflictions brought by complex timing constraints and further improve schedulability, this paper proposes the ROM and BPP procedures. The simulation results have shown that the proposed approaches significantly outperform ILP and prior peer heuristics for various settings.

REFERENCES

[1] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert, "Trends in Automotive Communication Systems," *Proc. IEEE*, vol. 93, pp. 1204-1224, 2005.
[2] CAN in automation. http://www.can-cia.org/can/
[3] TTCAN. http://www.can-cia.org/can/ttcan/.
[4] "The flexray communication system specification, version 3.0.1," http://www.flexray.com.
[5] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proc. IEEE*, vol. 91, no. 1, pp. 1120-126, 2003.
[6] M. Bertoluzzo, G. Buja, and A. Zuccollo, "Design of drive-by-wire communication network for an industrial vehicle," in *Proc. IEEE Int. Conf. Ind. Informat.*, pp. 155-160, Jun. 2004.
[7] J. Broy and K. Muller-Glaser, "The impact of time-triggered communication in automotive embedded systems," in *Proc. IEEE Symp. Ind. Embed. Syst.*, pp. 353-356, 2007.
[8] M. Short and M. Pont, "Fault-tolerant time-triggered communication using CAN," *IEEE Trans. Ind. Informat.*, vol. 3, no. 2, pp. 131-142, May 2007.
[9] K. C. Lee, M. H. Kim, S. Lee, and H. H. Lee, "IEEE-1451-based smart module for in-vehicle networking systems of intelligent vehicles," *IEEE Trans. Ind. Electron.*, vol. 51, no. 6, pp. 1150-1158, Dec. 2004.
[10] I. Park and M. Sunwoo, "FlexRay network parameter optimization method for automotive applications," *IEEE Trans. Ind. Electron.*, vol.58, no. 4, pp. 1449-1459, Apr., 2011.
[11] H. Zeng, , M.D. Natale, A. Ghosal, and A. Sangiovanni-Vincentelli. "Schedule optimization of time-triggered systems communicating over the FlexRay static segment," *IEEE Trans. Ind. Informat.*, vol. 7, no. 1, pp. 1-17, Jan. 2011.
[12] B. Tanasa, U.D. Bordoloi, P. Eles, and Z. Peng, "Scheduling for fault-tolerant communication on the static segment of FlexRay," *Proc. RTSS* 2010.
[13] M. Lukasiewycz, R. Schneider, D. Goswami, and S. Chakraborty, "Modular scheduling of distributed heterogeneous time-triggered automotive systems," *Proc. ASP-DAC*, 2012.
[14] T.C. Hu, "Parallel Sequencing and Assembly Line Problems," *Oper. Research*, vol. 19, no. 6, pp.841-848, Nov. 1961.
[15] J.J. Hwang, Y.C. Chow, F.D. Anger and C.Y. Lee, "Scheduling Precedence Graphs in Systems with Interprocessor Communication Times," *SIAM Journal of Computing*, vol. 18, no. 2, pp. 244-257, Apr. 1989.
[16] Y. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE Trans. Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506-521, May 1996.
[17] M. Wu and D. Gajski, "Hypertool: A Programming Aid for Message-Passing Systems," *IEEE Trans. Parallel and Distributed Systems*, vo. 1, no. 3, pp. 330-343, 1990.

[18] A. Davare, Q. Zhu, M.D. Natale, C. Pinello, S. Kanajan, and A.S. Vincentelli, "Period Optimization for Hard Real-time Distributed Automotive Systems," *Proceedings of the 44-th annual Design Automation Conference*, 2007.
[19] T. Pop, P. Eles, and Z. Peng, "Schedulability Analysis for Distributed Heterogeneous Time/Event Triggered Real-Time Systems," *Proceedings of 15th Euromicro Conference on Real-Time Systems*, 2003.
[20] P. Pop, K.H. Poulsen, V. Izosimov, and P. Eles, "Scheduling and Voltage Scaling for Energy/Reliability Trade-offs in Fault-Tolerant Time-Triggered Embedded Systems," In *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, 2007.
[21] H. Jaouani, R. Bouhouch, W. Najjar, and S. Hasnaoui, "Hybrid task and message scheduling in hard real time distributed systems over FlexRay bus," In *IEEE International Conference on Communications and Information Technology (ICCIT)*, pp. 21-26, 2012.
[22] D. Peng, K.G. Shin, and T.F. Abdelzaher, "Assignment and scheduling communicating periodic tasks in distributed real-time systems," *IEEE Trans. Software Engineering*, vol. 23, no. 12, Dec. 1997, pp. 745-758.
[23] T.F. Abdelzaher, and K.G. Shin, "Combined task and message scheduling in distributed real-time systems," Parallel and Distributed Systems, *IEEE Trans. Parallel and Distributed Systems*, vol. 10, no. 11, pp. 1179-1191, Nov., 1999.
[24] K. Ramamritham, "Allocation and Scheduling of Precedence-Related Periodic Tasks," *IEEE Trans. Parallel and Distributed Systems*, vol. 6, no. 4, pp. 412-420, Apr. 1995.
[25] T.F. Abdelzaher, and K.G. Shin. "Period-based load partitioning and assignment for large real-time applications," *IEEE Trans. Computers*, vol. 49, no. 1 , pp. 81-87, Jan. 2000.
[26] B.P. Dave, G. Lakshminarayana, and N.K. Jha, "COSYN: HardwareC-Software Co-Synthesis of Heterogeneous Distributed Embedded Systems," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 92-104, Jan. 1999.
[27] G. Manimaran, C. Murthy, "An efficient dynamic scheduling algorithm for multiprocessor real-time systems," *IEEE Trans. Parallel Distrib. Systems.* vol. 9 , pp. 312-319, Mar. 1998.
[28] E. Schmidt and K. Schmidt, "Message scheduling for the flexray protocol: The dynamic segment,"*IEEE Trans. Vehicular Technology*, vol. 58, no. 5, pp. 2160-2169, 2009.
[29] M. Lukasiewycz, M. Glab, J. Teich, and P. Milbredt, "Flexray schedule optimization of the static segment," *Proc. CODES+ISSS*, 2009.
[30] K. Schmidt and E. Schmidt, "Message scheduling for the flexray protocol: The static segment," *IEEE Trans. Vehicular Technology*, vol. 58, no. 5, pp. 2170-2179, 2009.
[31] S. Ding, N. Murakami, H. Tomiyama, and H. Takada, "A GA-Based Scheduling Method for FlexRay Systems," *In Proc. International Conference on Embedded Software*, 2005.
[32] M. Grenier, L. Havet, and N. Navet, "Configuring the Communication on FlexRay: the Case of the Static Segment," *In Proc. ECRTS*, 2008.
[33] IBM, ILOG CPLEX, http://www.ibm.com/software/, Version 12.2.

Menglan Hu received the B.E. degree in Electronic and Information Engineering from Huazhong University of Science and Technology, Wuhan, China, in 2007, and the Ph.D. degree in Electrical and Computer Engineering from the National University of Singapore, Singapore, in 2012. From 2012 to 2014 he was a research fellow at the School of Computer Engineering, Nanyang Technological University, Singapore. In 2014 he joined the faculty of the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, Wuhan, China, where he is currently an Associate Professor. His research interests includes cloud computing, parallel and distributed systems, scheduling and resource management, as well as wireless networking.

Jun Luo received the B.S. and M.S. degrees in electrical engineering from Tsinghua University, Beijing, China, in 1997 and 2000, respectively, and the Ph.D. degree in computer science from the Swiss Federal Institute of Technology in Lausanne (EPFL), Lausanne, Switzerland, in 2006. From 2006 to 2008, he has worked as a Post-Doctoral Research Fellow with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. In 2008, he joined the faculty of the School of Computer Engineering, Nanyang Technological University, Singapore, where he is currently an Assistant Professor. His research interests include wireless networking, mobile and pervasive computing, distributed systems, multimedia protocols, network modeling and performance analysis, applied operations research, and network security. He is a member of the IEEE.

Yang Wang received the BS degree in applied mathematics from the Ocean University of China in 1989 and the MS and PhD degrees in computing science from Carleton University and the University of Alberta in 2001 and 2008, respectively. He is currently at IBM Center for Advanced Studies (CAS), Atlantic, University of New Brunswick, Fredericton, Canada. Before joining CAS Atlantic in 2012, he was a research fellow at the National University of Singapore from 2010 to 2012. Before that, he was a research associate at the University of Alberta, Canada, from August 2008 to March 2009. His research interests include scientific workflow computation and virtualization in Clouds and resource management algorithms.

Martin Lukasiewycz received the Dipl.Ing. and Ph.D. degree in computer science from the University of Erlangen-Nuremberg, Germany, in 2006 and 2010, respectively. Since 2011, Martin Lukasiewycz works as Principal Investigator on automotive Embedded Systems at the TUM CREATE Centre for Electromobility in Singapore. From 2006 to 2010, he worked as research assistant at AUDI AG and the Department of Hardware-Software Co-Design, University of Erlangen-Nuremberg, before he joined the Institute for Real-Time Computer Systems, Technical University of Munich, as a postdoctoral researcher in 2010. His research covers automotive in-vehicle communication networks, system-level design of embedded systems, and meta-heuristic optimization. Dr. Lukasiewycz serves on the program committee of major conference such as DATE and DAC. He is a member of the IEEE.

Zeng Zeng received the Ph.D. in electrical and computer engineering from The National University of Singapore, in 2005. He received his BS and MS degrees in automatic control from Huazhong University of Science and Technology, Wuhan, P.R. China, in 1997 and 2000, respectively. His research interests include distributed/grid computing systems, multimedia storage systems, wireless sensor networks, and controller area networks. Currently, he works as a senior research fellow at The National University of Singapore and in the meanwhile, he is the founder of GoGoWise Cloud Education Pte. Ltd, Singapore.