06AE-258

The Synthesis of Dependable Communication Networks for Automotive Systems

Nagarajan Kandasamy
Drexel University, Philadelphia, USA

Fadi Aloul
American University of Sharjah, Sharjah, UAE

ABSTRACT

Embedded automotive applications such as drive-by-wire in cars require dependable interaction between various sensors, processors, and actuators. This paper addresses the design of low-cost communication networks guaranteeing to meet both the performance and fault-tolerance requirements of such distributed applications. We develop a fault-tolerant allocation and scheduling method which maps messages on to a low-cost multiple-bus system to ensure predictable inter-processor communication. The proposed method targets time-division multiple access (TDMA) communication protocols. Finally, we present a case study using some advanced automotive control applications to show that our approach uses the available network bandwidth efficiently to guarantee message deadlines.

INTRODUCTION

Embedded computers are being increasingly used in automobiles to replace safety-critical mechanical and hydraulic systems. Drive-by-wire is one example where traditional hydraulic steering and braking are replaced by a networked microprocessor-controlled electro-mechanical system [1]. Sensors measure the steering-wheel angle and brake-pedal position, and processors calculate the desired road-wheel and braking parameters which are then applied via electro-mechanical actuators at the wheels. Other computerized vehicle-control applications including adaptive cruise control, collision avoidance, and autonomous driving are also being developed [2]. These applications will be realized as *real-time distributed systems* requiring dependable and timely interaction between sensors, processors, and actuators. This paper addresses the design of low-cost communication networks to meet both the performance and fault-tolerance requirements of such applications.

The approach described in this paper synthesizes a fault-tolerant (FT) network topology from application requirements. While synthesis methods such as [3] assume an underlying CAN communication protocol and arbitrate bus access using message (processor) priorities, we target TDMA communication protocols where processors are allotted transmission slots according to a static, periodic, and global communication schedule [4]. Examples include TTP [5] and FlexRay [6] that have recently emerged as possible networking standards for in-vehicle networks.

We restrict the network topology space to multiple-bus systems such as the one in Fig. 1 where each processor P_i connects to a subset of the communication buses. A co-processor handles message communication independently without interfering with task execution on P_i. A multiple-bus topology allows fault-tolerant message allocation. Also, since communication protocols for the embedded systems of interest are typically implemented over low-cost physical media, individual buses have limited bandwidth. Therefore, multiple buses may be needed to accommodate the message load.

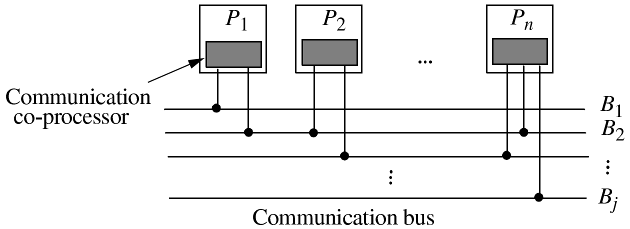


Fig. 1: A multi-bus system where each processor connects to a subset of the communication buses

Given a set of distributed applications modeled as task graphs $\{G_i\}$, the proposed approach generates a communication network satisfying both the performance and fault-tolerance requirements of each G_i. Messages are allocated and scheduled on the minimum number of buses $\{B_j\}$ where each B_j has a specified bandwidth. The major features of our approach are as follows:

- It assumes a multi-rate system where each graph G_i may have a different execution period *period*(G_i).
- It targets a TDMA communication protocol.

- It supports dependable message communication by establishing redundant transmission paths between processors, thereby tolerating a bounded number of permanent bus failures.
- It uses network bandwidth efficiently by reusing transmission slots allotted to a processor between the multiple messages sent by it.

Finally, using representative distributed automotive control applications, we show that the proposed method guarantees predictable message transmission while reducing bandwidth utilization.

The rest of this paper is organized as follows. Section 2 presents an overview of the proposed approach, while Section 3 discusses some preliminaries including task scheduling. The message allocation method is developed in Section 4, and Section 5 presents the case study. We conclude the paper in Section 6.

DESIGN FLOW

As the primary objective, we construct a network topology meeting the fault-tolerance and performance goals of the embedded applications. The secondary objective is to minimize hardware cost in terms of communication buses. An heuristic method is developed where a feasible network topology satisfying performance goals is first obtained. Its cost is then reduced via a series of steps which minimize the number of buses by appropriately grouping (clustering) messages while preserving the feasibility of the original solution.

The main steps of the proposed design approach are as follows. For a given allocation of task to processors $\{P_i\}$, the corresponding inter-processor messages are mapped to a low-cost network topology comprising identical buses $\{B_j\}$. Redundant routes are provided for messages with specific fault-tolerance requirements; for a k-fault-tolerant (k-FT) message m_i, k replicas or copies are allocated to separate buses. The network is synthesized assuming a generic TDMA protocol, and can be modified to accommodate specific cases such as TTP and FlexRay.

We assume that each task graph G_i must meet its deadline by the end of its period *period*(G_i). First, the graph deadline is distributed over its tasks to generate a scheduling range $[r_i, d_i]$ for each task T_i where r_i and d_i denote its release time and deadline, respectively. The initial network topology is obtained by simply allocating each inter-processor message m_i to a separate bus. Without bus contention, m_i's transmission delay is given by the message size and bus bandwidth, and the overall solution is feasible if all tasks complete before their respective deadlines. The next section discusses these preliminary steps in greater detail.

The number of communication buses in the initial solution is then minimized via an iterative message clustering procedure which groups multiple messages on bus B_j. A message m_i is grouped with an existing cluster C_j if the resulting allocation satisfies the following requirements: (1) No two replicas of a k-FT message are allocated to C_j. (2) All messages belonging to C_j continue to meet their deadlines. (3) The duration (length) of the communication schedule corresponding to C_j does not exceed a designer-specified threshold. Note that if a dedicated co-processor handles communication as in Fig. 1, the message transmission schedule must be compact enough to fit within the available memory.

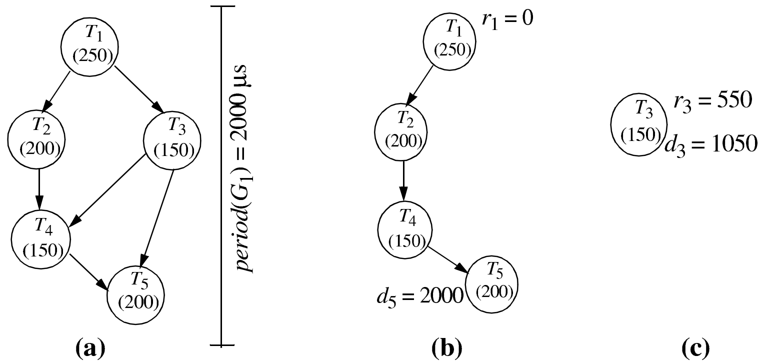
The proposed clustering approach also uses bus bandwidth efficiently by sharing or re-using transmission slots between multiple messages sent by a processor whenever possible. Each message cluster is allocated to a separate bus in the final topology.

PRELIMINARIES

This section shows how to obtain the initial solution where tasks are assigned deadlines and scheduled on processors, and messages allocated to separate communication buses.

Deadline Assignment: Initially, only the entry and exit tasks having no predecessors and successors, respectively, have their release times and deadlines fixed. To schedule an intermediate task T_i in the task graph, however, its scheduling range $[r_i, d_i]$ must first be obtained. This is termed the deadline assignment problem where the deadline D_i of the task graph G_i must be distributed over each intermediate task such that all tasks are feasibly scheduled on their respective processors. Deadline distribution is NP-complete and various heuristics have been proposed to solve it. We use the approach of [7] which maximizes the slack added to each task in graph G_i while still satisfying its deadline D_i.

We now describe the deadline distribution algorithm. Entry and exit tasks in the graph are first assigned release times and deadlines. A path $path_i$ through G_i comprises one or more tasks $\{T_i\}$; the slack available for distribution to these tasks is $slack_i = D_i - \sum c_i$ where D_i is the deadline of $path_i$ and c_i the execution time of a task T_i along this path. The distribution heuristic in [7] maximizes the minimum slack added to each T_i along $path_i$ by dividing $slack_i$ equally among tasks. During each iteration through G_i, $path_i$ minimizing $slack_i / n$, where n denotes the number of tasks along $path_i$, is chosen and the corresponding slack added to each task along that path. The deadlines (release times) of the predecessors (successors) of tasks belonging to $path_i$ are updated. Tasks along $path_i$ are then removed from the original graph, and the above process is repeated until all tasks are assigned release times and deadlines.



(a) **(b)** **(c)**

T_i	r_i (µs)	d_i (µs)
T_1	0	550
T_2	550	1050
T_3	550	1050
T_4	1050	1500
T_5	1500	2000

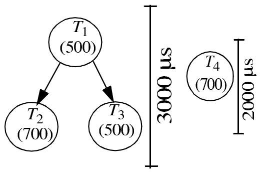
(d)

Fig. 2: (a) Example task graph; (b) and (c) paths selected for deadline distribution, and (d) the resulting scheduling ranges for each task

We use the graph in Fig. 2(a) to illustrate the above procedure. First, the release time of entry task T_1 and the deadline of exit task T_5 are set to $r_1 = 0$ µs and $d_5 = 2000$ µs, respectively. Next, we select the path $T_1T_2T_4T_5$ shown in Fig. 2(b); the total execution time of tasks along this path is 800 µs, and as per the heuristic, a slack of $(2000 - 800)/4 = 300$ µs is distributed to each task. Once their release times and deadlines are fixed, these tasks are removed from the graph. Fig. 2(c) shows the remaining path comprising task T_3 which has its release time and deadline fixed by T_1 and T_4, respectively. Fig. 2(d) shows the resulting scheduling range for each task.

Task Scheduling: Once the scheduling ranges of tasks in the graph are fixed, each T_i may now be considered independent with release time r_i and deadline d_i, and scheduled as such. To tackle multi-rate systems, we use *fixed-priority scheduling* where tasks are first assigned priorities according to their periods [8], and at any time instant, the processor executes the highest-priority ready task. Again, the schedule is feasible if all tasks finish before their deadlines. Feasibility analysis of schedules using simple closed-form processor-utilization-based tests has been extensively studied under fixed-priority scheduling. However, in addition to feasibility, we also require task T_i's response time w_i, given by the time interval between T_i's release and finish times; the response time is used in the next stage of our algorithm to determine the message delays to be satisfied by the network.

For multi-rate task graphs, the schedules on individual processors are simulated for duration equal to the least common multiple (LCM) of the graph periods. Since this duration evaluates all possible interactions between tasks belonging to the different graph iterations, the worst-case response time for each task T_i is obtained. Fig. 3(a) shows a simple multi-rate system comprising two task graphs with periods 2000 µs and 3000 µs; Figs. 3(b) and 3(c) show the task allocation and scheduling ranges, respectively. Fig. 3(d) shows the corresponding schedule for 6000 µs–the LCM of the graph periods. Task response times within this time interval are shown in Fig. 3(e). Multiple iterations of a task are evaluated to obtain its worst-case response time. For example, in Fig. 3(e), the first iteration of tasks T_1, T_2, and T_4 (in bold) has the maximum response time among the iterations within



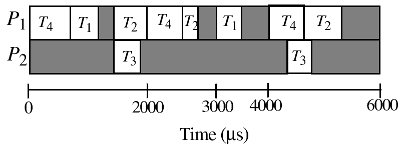
(a)

P_i	Tasks
P_1	T_1, T_2, T_4
P_2	T_3

(b)

T_i	r_i (µs)	d_i (µs)
T_1	0	1400
T_2	1400	3000
T_3	1400	3000
T_4	0	2000

(c)



(d)

T_i	Iteration	w_i (µs)
T_1	**1**	**1200**
	2	500
T_2	**1**	**1400**
	2	1000
T_4	**1**	**700**
	2	700
	3	700

(e)

Fig. 3: (a) An example multi-rate system, (b) task-to-processor allocation, (c) task scheduling ranges, (d) task schedule for the duration of the least common multiple of the task periods, and (e) the response times of different task iterations over the simulated time interval

the given time duration. The task scheduling on processors is successful if, for each task T_i, $w_i \le d_i - r_i$. However, for the overall solution to be feasible, all messages must also meet their deadlines.

Initial Network Topology: A *k*-FT message m_i sent by task T_i has deadline $delay(m_i) = d_i - r_i - w_i$ where w_i denotes T_i's worst-case response time. Initially, the network topology allocates a separate communication bus for each message copy. Therefore, in this topology, m_i experiences no network contention and its transmission delay is $size(m_i) \big/ B_j^{\text{speed}}$ where *size*(m_i) and B_j^{speed} denote the message size in bits and bus bandwidth in kb/s, respectively. The solution is feasible if, for each m_i, *delay*(m_i) is greater than the corresponding transmission delay.

MESSAGE CLUSTERING

We now develop a clustering approach to reduce the cost of the initial network topology where multiple messages are grouped on a single bus while preserving the feasibility of the original solution. The fault-tolerance requirement of each *k*-FT message is also satisfied during this procedure.

First, we briefly review message transmission in a typical TDMA communication protocol such as FlexRay. Messages are transmitted according to a static, periodic, and global communication schedule called a *round,* comprising identical-sized slots. Each processor P_j is allotted one or more sending slots during a round where both slot size and the number of slots per round are fixed by the system designer. Though successive rounds are constructed identically, the messages sent by processors may vary during a given round.

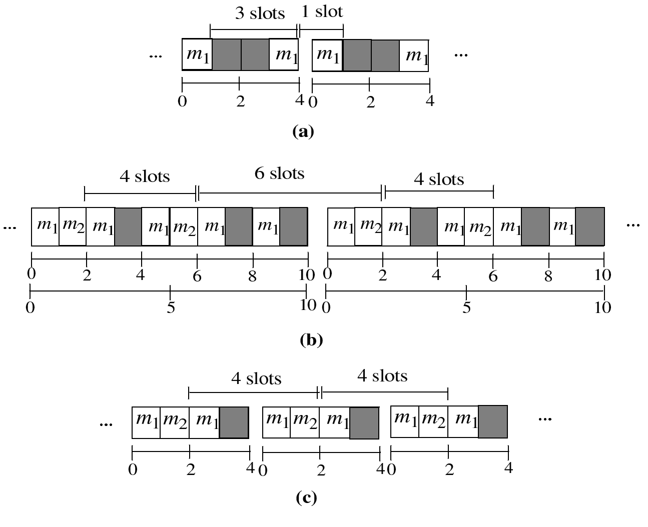


Fig. 4: (a) Message allocation resulting in a missed deadline; (b) a clustering of multiple messages resulting in missed deadlines, and (c) a clustering guaranteeing deadlines, obtained after modifying message periods appropriately

We now state the fault-tolerant message clustering problem as follows. Given a communication deadline *delay*(m_i) for each *k*-FT message m_i sent by processor P_j, construct TDMA rounds on the minimum number of communication buses such that during any time interval corresponding to *delay*(m_i), P_j is allotted a sufficient number of transmission slots to transmit m_i. Allocation of messages to multiple buses is related to *bin-packing* where messages are packed into a bin (round) of finite size while minimizing the number of bins. The general bin-packing problem is NP-complete and heuristics are typically used to obtain a solution [9].

We treat each m_i as a periodic message with period *period*(m_i) equal to its deadline *delay*(m_i) and generate message clusters $\{C_j\}$, such that the corresponding TDMA round *round*(C_j) satisfies the following constraints: (1) No two replicas of a *k*-FT message m_i are allocated to C_j. (2) The duration of *round*(C_j) does not exceed a designer-specified threshold. (3) The slots within *round*(C_j) guarantee m_i's deadline, i.e., the time interval between successive sending slots for m_i equals its period.

Each message cluster C_j is allocated to a separate communication bus in the final network topology. Our method also makes efficient use of bus bandwidth by minimizing the number of transmission slots needed to satisfy message deadlines within a TDMA round by reusing slots between messages sent by a processor whenever possible.

We assume an upper bound on TDMA-round duration provided by the designer in terms of the maximum number of transmission slots $n_{\max}$ and slot duration Δ_{slot}. Typically, the choice of $n_{\max}$ depends on the memory limitations of the communication co-processor such as the number of transmit and receive buffers. Each transmission slot within a round has duration $\Delta_{\text{slot}} = \min_i\{size(m_i)\} \big/ B_j^{\text{speed}}$ μs. The message period *delay*(m_i), originally expressed in μs, is now discretized as $\lfloor delay(m_i)/\Delta_{\text{slot}} \rfloor$ and expressed in terms of transmission-slot intervals. To simplify the notation, we will use *delay*(m_i) to denote this discrete quantity from here on.

To guarantee message m_i's deadline, the corresponding slot allocation must satisfy both its periodicity requirement and a distance constraint between successive m_i transmissions as the following example illustrates. Fig. 4(a) shows an allocation scenario for message m_1 having *delay*(m_1) = 2 slots within a TDMA round of duration four slots where m_1 requires one slot for transmission. Though m_1's periodicity requirement may be satisfied by simply allocating sufficient slots within each of its periods, it results in missed deadlines. The interval between successive m_1 transmissions may be as close to one and as far as three slots away. As Fig.

```
Procedure SYNTH ({$m_i$})                 /* {$m_i$} := Message set */
    $p_{\min} = \min_i\{period(m_i)\}$;            /* $p_{\min}$ denotes the minimum period in the message set */
    $cost_{\min}$ := Number of messages in {$m_i$}; /* Topology cost where each $m_i$ is allotted a dedicated bus */
    $s_{msg} := \varnothing$;                     /* Initialize the message set */
    for (each $p_{base}$ in [$\frac{p_{\min}}{2}$, $p_{\min}$]) begin
        $s_{msg}$ := {$m_i$ | $m_i$'s period is the largest integer such that $2^k \cdot p_{base} \le delay(m_i) < 2^{k+1} \cdot p_{base}$};
        Sort messages in $s_{msg}$ by increasing period;
        $s_{clust}$ := CLUSTER($s_{msg}$);       /* $s_{clust}$:= set of clusters */
        $cost_{cur}$ := Number of clusters in $s_{clust}$;
        if ($cost_{cur} < cost_{\min}$) begin
            $cost_{\min} := cost_{cur}$;
            Store $s_{clust}$ as current best solution;
        end;
        $s_{msg} := \varnothing$;
    end;
    return $s_{clust}$;      /* Return the best allocation */
```

Fig. 5: Algorithm to synthesize the network topology

4(a) shows, in the worst case, m_1 may be allocated a transmission slot just before the end of its current period and one immediately at the start of its next period. Clearly, this results in a deadline violation. Similar problems may also occur when multiple messages are clustered.

Figure 4(b) shows TDMA rounds corresponding to messages m_1 and m_2 with periods *period*(m_1) = 2 and *period*(m_2) = 5 slots, respectively. Transmission slots are allocated in *first-fit* (FF) fashion where messages are ordered in terms of increasing period and the first available slots allocated to each m_i within the round. The slot allocation scheme in Fig. 4(b) results in a deadline violation where the minimum and maximum distances between successive slots for m_2 are four and six slots, respectively. Therefore, to guarantee message m_i's deadline, the corresponding allocation must satisfy a maximum distance between successive m_i transmission slots equal to *period*(m_i). Note that in the above example, message deadlines may be satisfied by modifying their periods appropriately. Fig. 4(c) shows the slot allocation for both messages after m_2's period is modified to four slots. It is easily checked that the distance constraint of two and four slots for successive transmissions of m_1 and m_2, respectively, is satisfied.

The above discussion suggests that the original message periods may need modification prior to allocating slots within the TDMA round. We adopt a strategy where the message periods within a cluster are constrained to be harmonic multiples of some base period p_{base}, i.e., $period(m_i) = 2^k \cdot p_{base}$, a concept used when scheduling tasks in real-time systems requiring a specific temporal separation between successive task executions [10]. We constrain each m_i's period to be the maximum integer $period(m_i) \le n_{\max}$ satisfying:

$$2^k \cdot p_{base} \le delay(m_i) < 2^{k+1} \cdot p_{base}$$

If $p_{\min} = \min_i\{period(m_i)\}$ denotes the smallest period among the messages, then $p_{\min}/2 < p_{base} \le p_{\min}$. Fig. 5 shows the synthesis algorithm to construct the network topology. For each p_{base} value between [$p_{\min}/2$, $p_{\min}$], message periods are modified appropriately, and clustered to generate the corresponding topology. Finally, the best solution, in terms of the number of clusters, is chosen.

```
Procedure ALLOC ($C_j$, $m_i$) /* $C_j$ := Message cluster; $m_i$ := Message */
    $s$ := Set of messages $\{C_j \cup m_i\}$ sorted in increasing period order;
    Create an empty TDMA round with $p_{\max} = \max_i\{period(m_i)\}$ slots;
    while ($s \ne \varnothing$) begin
        $m_i$ := Message with minimum period in $s$;
        $k = p_{\max}/period(m_i)$;        /* $k$ := Number of intervals */
        Divide the TDMA round into $k$ intervals $\{I_k\}$, each of duration $period(m_i)$;
        $n := \lceil size(m_i)/\Delta_{slot} \rceil$;  /* Number of slots needed to accommodate $m_i$ */
        for (each interval $I_k$) begin
            if ($n$ free slots are unavailable) return $\varnothing$; /* Allocation is infeasible */
            Allocate $n$ slots within $I_k$ to $m_i$ in first-fit (FF) fashion;
        end;
        $s := s - \{m_i\}$;
    end;
    return TDMA round; /* Return the feasible allocation */
```

```
Procedure CLUSTER($s_{msg}$) /* $s_{msg}$ := Messages {$m_i$} sorted by increasing period */
    $s_{clust} := \varnothing$;                  /* Initialize set of message clusters */
    while ($s_{msg} \ne \varnothing$) begin
        $m_i$ := $k$-FT message in $s_{msg}$ with minimum period;
        $s_{cand} := \varnothing$;               /* Initialize set of possible candidate clusters */
        for (each compatible cluster $C_j$ in $s_{clust}$) /* Allocate $k$-FT message to clusters */
            if (ALLOC($C_j$, $m_i$) returns a feasible TDMA round) $s_{cand} := s_{cand} \cup C_j$;
        $n_{cand}$ := Number of clusters in set $s_{cand}$;
        if ($n_{cand} < k$) begin /* New clusters are needed to accommodate copies of $m_i$ */
            $s_{clust} := s_{clust} \cup S_{cand}$;
            Allocate $m_i$ to ($k - n_{cand}$) new clusters and add them to $s_{clust}$;
        end;
        if ($n_{cand} \ge k$) begin      /* Select the best $k$ clusters in terms of slot reuse */
            Sort clusters in $S_{cand}$ in terms of decreasing slot reuse;
            Select the first $k$ clusters in the sorted set $S_{cand}$ and add to $S_{clust}$;
            Remove $m_i$ from the non-selected clusters;
        end;
        $S_{msg} := S_{msg} - m_i$;
    end;
    return $S_{clust}$;                   /* Output the set of message clusters */
```

Fig. 6: The clustering algorithm generating the reduced-cost network topology

The CLUSTER procedure shown in Fig. 6 takes a set of messages s_{msg} as input, their periods modified and sorted in terms of increasing *period*(m_i), and returns the set of message clusters s_{clust} as output. During each clustering step, we choose a *k*-FT message m_i having the minimum period within s_{msg} and allocate it to *k* separate clusters. For each m_i, we obtain all feasible message-to-cluster allocations by grouping m_i with each C_j in s_{clust} and generating *round*($C_j \cup m_i$). If needed, new clusters are created within s_{clust} to accommodate all copies of m_i. If more than *k* feasible allocations are obtained, then the *k* best solutions are chosen based on efficient bandwidth use.

The ALLOC procedure generates a feasible *round*($C_j \cup m_i$). It accepts an existing message cluster C_j and a message m_i and generates a feasible TDMA round (if possible) for the new allocation $C_j \cup m_i$. As discussed above, message m_i's period *period*(m_i) is first transformed to relate harmonically to those in C_j and the messages are sorted in increasing period order. The duration of the new round *round*($C_j \cup m_i$) is $p_{\max} = \max_{C_j}\{period(m_i)\}$. To allocate transmission slots for the new message m_i, ALLOC divides *round*(C_j) into *k* disjoint time intervals $\{I_k\}$ where $k = p_{\max}/period(m_i)$ and I_k has duration *period*(m_i). Transmission slots are then allotted within each interval

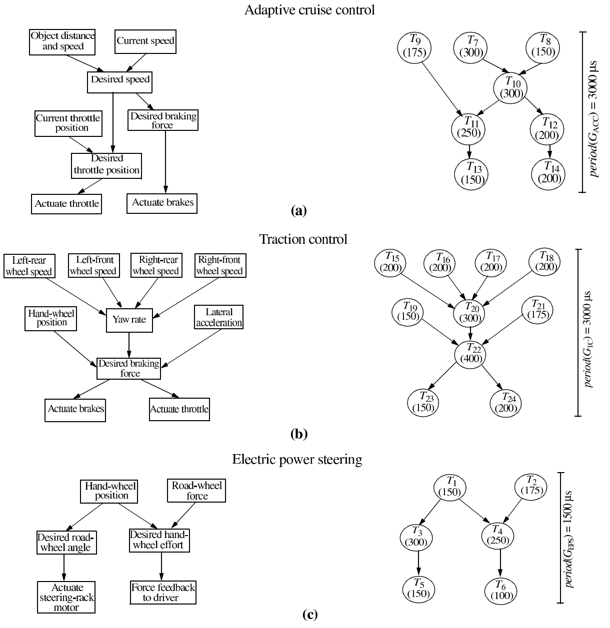
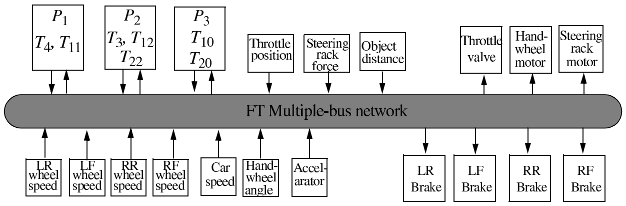


Fig. 7: (a) Adaptive cruise control, (b) traction control, and (c) electric power steering applications, and the corresponding flow-graph representations

using the FF packing strategy. The distance constraint between transmission slots for m_i is guaranteed since the allotted slots occur in the same positions within each interval I_k.

Transmission-Slot Reuse: Recall that during clustering, each message m_i is treated as periodic with period *period*(m_i). However, if the task T_i transmitting m_i does not execute at that rate, then the bus bandwidth is over-utilized. We can improve bandwidth utilization by reusing transmission slots among the multiple messages sent by processor P_j.

The worst-case arrival rate *arrival*(m_i) for each message m_i in a multi-rate system is obtained during schedulability analysis by simulating the corresponding task schedule. It is important to note that *arrival*(m_i), expressed in terms of slot intervals, depends on the execution rate of the sender task T_i. Let $\{m_i\}$ be the set of messages sent by a processor within a message cluster C_j. Now, assume message m_{new}, also transmitted by the same processor, to be allotted slots within *round*(C_j). If each message m_i is allotted n_i transmission slots within the time interval *period*(m_{new}) in *round*(C_j), then the number of slots available for reuse by m_{new} is



(a)

Message m_i	(Sender, receiver)	$size(m_i)$ (bits)	$delay(m_i)$ (μs)	$delay(m_i)$ (slot intervals)
m_1	(T_1, T_3) (T_1, T_4)	12	300	6
m_2	(T_2, T_4)	12	275	5
m_3	(T_3, T_5)	8	300	6
m_4	(T_4, T_6)	12	350	7
m_5	(T_7, T_{10})	12	500	10
m_6	(T_8, T_{10})	12	650	6
m_7	(T_9, T_{11})	10	1425	28
m_8	(T_{10}, T_{11}) (T_{10}, T_{12})	12	500	10
m_9	(T_{11}, T_{13})	10	500	10
m_{10}	(T_{12}, T_{14})	10	500	10
m_{11}	(T_{15}, T_{20})	12	475	9
m_{12}	(T_{16}, T_{20})	12	475	9
m_{13}	(T_{17}, T_{20})	12	475	9
m_{14}	(T_{18}, T_{20})	12	475	9
m_{15}	(T_{19}, T_{22})	10	1100	22
m_{16}	(T_{20}, T_{22})	10	275	5
m_{17}	(T_{21}, T_{22})	8	1025	20
m_{18}	(T_{22}, T_{23}) (T_{22}, T_{24})	12	650	6

(b)

Fig. 8: (a) The physical architecture including task-to-processor allocation and (b) the message attributes required for network topology

$$n_{\text{reuse}} = \sum_{m_i} n_i - \sum_{m_i} \left\lceil \frac{period(m_{\text{new}})}{arrival(m_i)} \right\rceil \cdot n_i$$

where *arrival*(m_i) denotes the worst-case arrival rate of message m_i. Therefore, m_{new} is allotted $\left\lceil \frac{size(m_{\text{new}})}{B_j^{\text{speed}} \cdot \Delta_{\text{slot}}} \right\rceil - n_{\text{reuse}}$ transmission slots within *period*(m_{new}).

Given a set of clusters and a new message to be allocated to one, CLUSTER explores all possible cluster-message allocation scenarios. Slot reuse is used as the deciding factor in selecting the best allocation since the cluster allocation resulting in maximum reuse minimizes the bandwidth utilization. Finally, when TDMA slots are shared between messages sent by a processor, the communication co-processor must correctly schedule their transmission, i.e., given a slot, decide which message to transmit in it. Though this paper does not address message-scheduling logic within the co-processor, an earliest-deadline first approach seems appropriate.

Bus attribute	Value
B_i^{speed} (Bus speed)	250 kb/s
Δ_{slot} (Slot size)	50 µs
$n_{\max}$ (Max. number of slots)	16

(a)

p_{base} = 3 slots

12 slots

B_1	m_2	m_{16}	m_1	m_2	m_{16}	m_3	m_2	m_{16}	m_1	m_2	m_{16}	m_3
B_2	m_4	m_5	m_6	m_8	m_9	m_{10}	m_4	m_5	m_6	m_8	m_9	m_{10}
B_3	m_{11}	m_{12}	m_{13}	m_{14}	m_{18}	m_7	m_{11}	m_{12}	m_{13}	m_{14}	m_{18}	m_{15}
B_4	m_{17}											

(b)

p_{base} = 4 slots

16 slots

B_1	m_1	m_2	m_3	m_4	m_1	m_2	m_3	m_4	m_1	m_2	m_3	m_4	m_1	m_2	m_3	m_4
B_2	m_6	m_{16}	m_{18}	m_5	m_6	m_{16}	m_{18}	m_8	m_6	m_{16}	m_{18}	m_5	m_6	m_{16}	m_{18}	m_8
B_3	m_9	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}	m_7	m_{15}	m_9	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}	m_{17}	

(c)

p_{base} = 5 slots

10 slots

B_1	m_1	m_2	m_3	m_4	m_6	m_1	m_2	m_3	m_4	m_6
B_2	m_{11}	m_{12}	m_{13}	m_{14}	m_{16}	m_{11}	m_{12}	m_{13}	m_{14}	m_{16}
B_3	m_{18}	m_7	m_8	m_5	m_9	m_{18}	m_{10}	m_{15}	m_{17}	

(d)

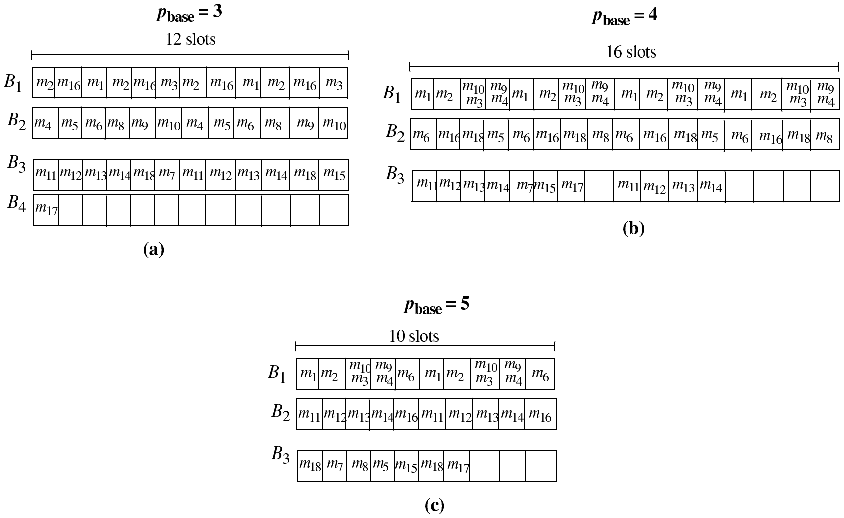
Fig. 9: (a) Example TDMA round specifications and (b) communication schedules generated without slot reuse where message periods are modified to relate harmonically to (b) p_{base} = 3, (c) p_{base} = 4, and (d) p_{base} = 5 slots, respectively

CASE STUDY

We now illustrate the proposed synthesis method using some advanced automotive control applications as examples. These include adaptive cruise control (ACC), electric power steering (EPS), and traction control (TC), and are detailed in Figs. 7(a)-(c). The ACC application automatically maintains a safe following distance between two cars, while EPS uses an electric motor to provide necessary steering assistance to the driver. The TC application actively stabilizes the vehicle to maintain its intended path even under slippery road conditions. These applications demand timely interaction between distributed sensors, processors, and actuators, i.e., have specific end-to-end deadlines, and therefore require a dependable communication network. Fig. 8(a) shows the physical architecture of the system where sensors and actuators are directly connected to the network and the designer-specified task-to-processor allocation, while Fig. 8(b) summarizes the various message attributes affecting network topology generation. We assume 1-FT messages throughout. Columns two and three list the sending and receiving tasks for each message and the message size *size*(m_i) in bits, respectively, while columns four and five list the communication delay *delay*(m_i) for messages in µs, and transmission-slot intervals. These delay values are obtained by first assigning deadlines to tasks and then performing a schedulability analysis on their respective processors.

As summarized in Fig. 9(a), we assume a version of the FlexRay communication protocol having a bandwidth of 250 kb/s and a minimum transmission-slot width of 50 µs. Since m_2 and m_{16} have the minimum period of five slots among all messages, p_{base} may assume values of three, four, or five slots. Figs. 9(a)-(c) show the communication schedules generated without slot reuse after modifying the message periods to relate harmonically to each of the above p_{base} values. Those corresponding to p_{base} values of four and five slots compare best in terms of topology cost.

We now show how to reduce bandwidth utilization by sharing transmission slots between messages. As candidates for slot reuse, consider messages m_3 and m_{10} sent by tasks T_3 and T_{12}, respectively, where both tasks are allocated to processor P_2. In Fig. 10(a), where message periods are modified using p_{base} = 3, m_3 and m_{10} cannot share slots since both have a periodicity of six slots. In Fig. 10(b), however, when their periods are modified as *period*(m_3) = 4 and *period*(m_{10}) = 8 using p_{base} = 4 slots, reuse is possible. Note that the EPS application comprising T_3 transmitting m_3 has a 1500 µs period-corresponding to the inter-interval time between successive m_3 transmissions. Therefore, in Fig. 10(b), m_3 requires only one of four allocated slots on bus B_1 (Task T_3, however, may request m_3's transmission anytime during the round), and m_{10} with a period of eight slots can reuse the one free slot available during any



$p_{base} = 3$

12 slots

B_1	m_2	m_{16}	m_1	m_2	m_{16}	m_3	m_2	m_{16}	m_1	m_2	m_{16}	m_3
B_2	m_4	m_5	m_6	m_8	m_9	m_{10}	m_4	m_5	m_6	m_8	m_9	m_{10}
B_3	m_{11}	m_{12}	m_{13}	m_{14}	m_{18}	m_7	m_{11}	m_{12}	m_{13}	m_{14}	m_{18}	m_{15}
B_4	m_{17}											

(a)

$p_{base} = 4$

16 slots

B_1	m_1	m_2	m_{10} m_3	m_9 m_4	m_1	m_2	m_{10} m_3	m_9 m_4	m_1	m_2	m_{10} m_3	m_9 m_4	m_1	m_2	m_{10} m_3	m_9 m_4
B_2	m_6	m_{16}	m_{18}	m_5	m_6	m_{16}	m_{18}	m_8	m_6	m_{16}	m_{18}	m_5	m_6	m_{16}	m_{18}	m_8
B_3	m_{11}	m_{12}	m_{13}	m_{14}	m_7	m_{15}	m_{17}		m_{11}	m_{12}	m_{13}	m_{14}				

(b)

$p_{base} = 5$

10 slots

B_1	m_1	m_2	m_{10} m_3	m_9 m_4	m_6	m_1	m_2	m_{10} m_3	m_9 m_4	m_6
B_2	m_{11}	m_{12}	m_{13}	m_{14}	m_{16}	m_{11}	m_{12}	m_{13}	m_{14}	m_{16}
B_3	m_{18}	m_7	m_8	m_5	m_{15}	m_{18}	m_{17}			

(c)

Fig. 10: (a) Communication schedules generated while reusing transmission slots for different values of p_{base}; (a) p_{base} = 3, (b) p_{base} = 4, and (c) p_{base} = 5 slots

period(m_{10}). A similar argument holds for messages m_4 and m_9 sent by processor P_1. Fig. 10(c) shows the schedule corresponding to p_{base} = 5 slots. Again, slots are reused between messages $\{m_3, m_{10}\}$ and $\{m_4, m_9\}$.

Finally, though the topologies shown in Figs. 10(b) and (c) have the same cost (three buses each), Fig. 10(b) has a somewhat lower slot utilization of 89.5% compared to 90% for Fig. 10(c). Since the empty slots in Fig. 10(b) may be used to transmit additional (non-critical) messages when compared to Fig. 10(c), we select the topology in Fig. 10(b) as the final solution.

CONCLUSION

This paper has addressed the synthesis of low-cost TDMA communication networks for distributed embedded systems. We have developed a fault-tolerant clustering method which allocates and schedules *k*-FT messages on the minimum number of buses to provide dependable transmission. The proposed method was illustrated using a case study involving some advanced automotive control applications and it was shown that sharing transmission slots among multiple messages reduces bandwidth consumption while preserving predictable communication. Therefore, the method has the potential to reduce topology cost when applied to larger embedded systems.

ACKNOWLEDGMENTS

This research was supported in part by a contract from The Delphi Corporation. Portions of this work appeared in SAFECOMP 2003, and the Journal of Reliability Eng., and System Safety, 2005.

REFERENCES

1. E. A. Bretz EA, “By-wire cars turn the corner,” *IEEE Spectrum*, vol. 38, no. 4, pp. 68-73, 2001.
2. G. Leen and D. Heffernan, “Expanding automotive electronic systems,” IEEE Computer, vol. 35, no. 1, pp. 88-93, 2002.
3. T. Y. Yen and W. Wolf, “Communication synthesis for distributed embedded systems,” *Proc. Int’l Conf. Computer-Aided Design*, pp. 288-294, 1995.
4. H. Kopetz, *Real-time systems: Design principles for distributed embedded applications*, Kluwer Academic Publishers, Boston, 1997.
5. H. Kopetz, “TTP: A time-triggered protocol for fault-tolerant real-time systems,” *Proc. IEEE Fault-Tolerant Computing Symp.*, pp. 524-533, 1993.
6. J. Berwanger et al., “FlexRay: The communication system for advanced automotive control systems,” Proc. SAE World Congress, Paper: 2001-01-0676, 2001.
7. M. D. Natale and J. A. Stankovic, “Dynamic end-to-end guarantees in distributed real-time systems,” *Proc. Real-Time Systems Symp.*, pp. 216-227, 1994.
8. X. Hu, J. G. D’Ambrosio, B. T. Murray, and D. L. Tang, “Co-design of architectures for automotive powertrain modules,” *IEEE Micro*, vol. 14, no. 4, pp. 17-25, 1994.
9. D. S. Johnson, “Fast algorithms for bin packing,” *J. Computer & System Sciences*, vol. 3, no. 3, pp. 272-314, 1974.
10. K. J. Lin and A. Herkert, “Jitter control in time triggered systems,” Proc. Hawaii Conf. System Sciences, pp. 451-455, 1996.