# Dependable Communication Synthesis for Distributed Embedded Systems*

Nagarajan Kandasamy[1], John P. Hayes[2], and Brian T. Murray[3]

[1]Institute for Software Integrated Systems, Vanderbilt University, Nashville, Tennessee, U.S.A
[2]Advanced Computer Architecture Lab., University of Michigan, Ann Arbor, Michigan, U.S.A
[3]The Delphi Corporation, Brighton, Michigan, U.S.A

**Abstract.** Embedded control 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 minimum-cost multiple-bus system to ensure predictable inter-processor communication. The proposed method targets time-division multiple access (TDMA) communication protocols, and is applicable to protocols such as FlexRay and TTP which have recently emerged as networking standards for embedded systems such as automobile controllers. Finally, we present a case study involving some advanced automotive control applications to show that our approach uses the available network bandwidth efficiently to achieve jitter-free message transmission.

## 1 Introduction

Embedded computer systems are being increasingly used in cost-sensitive consumer products such as automobiles to replace safety-critical mechanical and hydraulic systems [2]. 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. These applications will be realized as real-time distributed systems requiring dependable 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.

Related work in communication synthesis for distributed embedded systems belongs in two broad categories–those that assume a fixed network topology and schedule messages to meet deadlines [3] [4] [5], and those that synthesize a topology satisfying message deadlines [6] [8]. Ortega and Boriello [3] assume a fixed network topology using the controller area network (CAN) protocol and schedule messages by assigning appropriate priorities to help meet their deadlines. Abdelzaher and Shin [4] present an

*This research was supported by a contract from The Delphi Corporation.

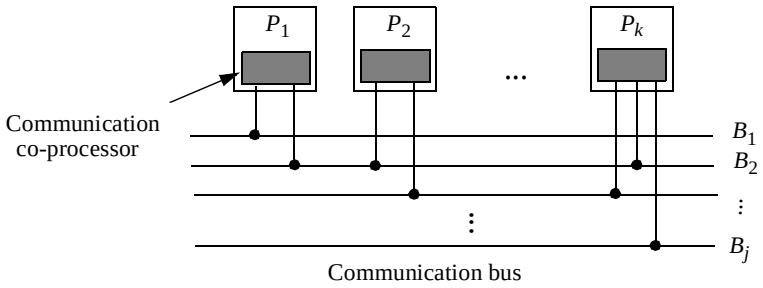



**Fig. 1. An example multiple-bus system where each processor connects to a subset of the communication buses**

off-line algorithm which schedules both tasks and messages in combined fashion to minimize the overall schedule length. Rhodes and Wolf [7] assign priorities to processors and schedule messages on a single communication bus using fixed priority, or round-robin arbitration for network access. A network topology satisfying message deadlines can also be constructed from application requirements. Given task graphs corresponding to embedded applications, Yen and Wolf [6] estimate the communication delay for inter-processor messages and schedule them on the minimum number of buses, while [8] generates point-to-point communication links.

Unlike [3] [4] [5] which assume a given topology, the approach proposed in this paper synthesizes the network topology from application requirements. Moreover, while synthesis methods such as [6] 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 [9]. Recently, TDMA protocols such as TTP [10] and FlexRay [11] have emerged as possible networking standards for an important class of embedded systems–automobiles.

Rather than generate arbitrary networks, we restrict the topology space to multiple-bus systems. Figure 1 shows an example where each processor $P_i$ connects to a subset of the communication buses. A co-processor handles message communication 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; multiple buses may be needed to accommodate the message load.

Given a set of distributed applications represented by task graphs $\{G_i\}$, our approach constructs a low-cost communication network that satisfies the performance and fault-tolerance requirements of each $G_i$. Messages are allocated and scheduled on the minimum number of buses $\{B_i\}$ where each $B_i$ has a specified bandwidth. We now summarize the major features of our approach:

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

```
Procedure FT-DESIGN({G_i}, {P_i}) /* {G_i} := Task graphs, {P_i} := Processors */
    for (each G_i)
        Distribute G_i's deadline to obtain the scheduling range [r_i, d_i] for each task T_i;
    for (each k-FT message m_i) begin      /* Obtain the initial network topology */
        Determine m_i's transmission delay tdelay(m_i);
        Allocate each copy of m_i to a separate bus B_j;
    end;
    for (each task T_i) begin              /* Determine task schedulability */
        w_i := Worst-case response time of T_i on its allocated processor P_i;
        if (w_i + tdelay(m_i) > d_i – r_i) return ∅; /* Solution is infeasible */
    end;
    s := CLUSTER ({m_i});                  /* Reduce topology cost via message clustering */
    Allocate each cluster C_i in s to a separate bus B_j;
    return {B_j};                          /* Return the set of communication buses */
```

**Fig. 2. The overall approach to fault-tolerant communication network synthesis**

- It supports fault-tolerant message communication by establishing redundant transmission paths between processors.

Finally, using some representative automotive control applications, we show that the proposed method guarantees jitter-free and predictable message transmission.

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

## 2 Design Overview

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 iterative 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. Since clustering is an NP-complete problem [12], we use heuristics to obtain a feasible solution.

Figure 2 shows the main steps of the proposed heuristic approach. For a given allocation of tasks to processors, FT-DESIGN accepts a set of task graphs $\{G_i\}$ and processors $\{P_i\}$ as inputs, and returns as output, a low-cost network topology comprising identical buses $\{B_i\}$. 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 designed assuming a generic TDMA protocol, and can accommodate specific cases such as TTP and FlexRay after some modification.

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 schedul-

ing 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 message $m_i$ to a separate bus. Without bus contention, $m_i$'s transmission delay is given by the message size and bus bandwidth. The overall solution is feasible if all tasks complete before their respective deadlines. Section 3 discusses these initial steps in greater detail.

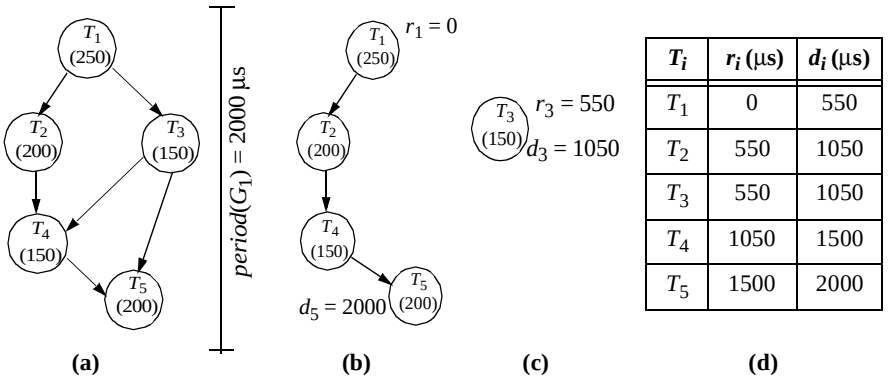
The number of communication buses in the initial solution is then minimized via an iterative message clustering procedure which groups multiple messages on bus $B_i$. A message $m_i$ is grouped with an existing cluster $C_j = \{m_i\}$ if the resulting communication schedule satisfies the following requirements: (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; if a dedicated co-processor handles message communication as in Fig. 1, the schedule must be compact enough to fit within the available memory. (4) The schedule provides jitter-free message transmission, where *jitter* is the uncertainty in the time intervals between successive transmissions of a message $m_i$. The proposed clustering approach also uses bus bandwidth efficiently by sharing or re-using transmission slots between multiple messages whenever possible. Each message cluster is allocated to a separate bus in the final topology. Section 4 describes this procedure in greater detail.

## 3 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 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 Natale and Stankovic [14] which maximizes the slack added to each task in graph $G_i$ while still satisfying its deadline $D_i$. Their heuristic is simple, and for general task graphs, its performance compares favourably with other heuristics [13].

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 [14] 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 $path_i$




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

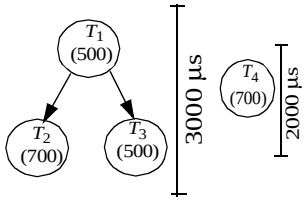
**Fig. 3. (a) Example task graph; (b) and (c) paths selected for deadline distribution, and (d) the resulting scheduling ranges for each task**

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.

We use the graph in Fig. 3(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. 3(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. Figure 3(c) shows the remaining path comprising only task $T_3$–it has its release time and deadline fixed by $T_1$ and $T_4$, respectively. Figure 3(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 [15], 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 [15]. However, in addition to feasibility, we also require a precise estimate of 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 a duration equal to the least common multiple (LCM) of the graph periods [16]. 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. Figure 4(a) shows a simple multi-rate system comprising two task graphs with periods 2000 µs and 3000 µs; Figs. 4(b) and 4(c) show the task allocation and scheduling ranges, respectively. Figure 4(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. 4(e). Multiple



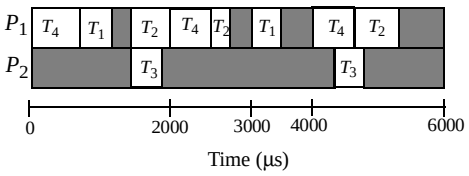

(a)

| $P_i$ | 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 | $w_i$ |
|---|---|---|
| $T_1$ | **1** | **1200** |
| | 2 | 500 |
| $T_2$ | **1** | **1400** |
| | 2 | 1000 |
| $T_4$ | **1** | **700** |
| | 2 | 700 |
| | 3 | 700 |

(e)

**Fig. 4. (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**

iterations of a task are evaluated to obtain its worst-case response time. For example, in Fig. 4(e), the first iteration of tasks $T_1$, $T_2$, and $T_4$ (in bold) has the maximum response time among the iterations within 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 $size(m_i) \times B_j^{speed}$ 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.

## 4 Fault-Tolerant Message Clustering

We now develop a message clustering approach to reduce the cost of the initial network topology obtained in Section 3. 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.

First, we briefly review message transmission in a generic TDMA communication protocol. As an example, we choose the FlexRay protocol currently under development by a consortium of automotive companies to provide predictable and high speed message communication for distributed control applications [11]. Figure 5 shows a a typi-

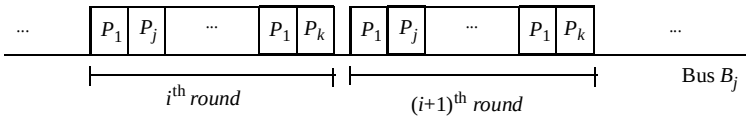



**Fig. 5. A TDMA-based allocation of transmission slots to processors on communication bus $B_j$**

cal TDMA scheme where messages are transmitted according to a static, periodic, and global communication schedule called a *TDMA round*. Each processor $P_i$ is allotted one or more sending slots during a round comprising a fixed number of identical-sized slots–both size and number of slots per round are fixed by the system designer and determine the round duration or period. Though successive rounds are constructed identically, the messages sent by individual processors may vary during a given round.

**Clustering Algorithm.** We now state the fault-tolerant message clustering problem as follows. Given the 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 $m_i$.

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 constraints previously introduced in Section 2: (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 $round(C_j)$ provide jitter-free message transmission, i.e., the time interval between successive sending slots for a message $m_i$ 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. This is achieved by reusing slots among the messages sent by a processor whenever possible. The following discussion describes the clustering procedure in greater detail. We assume an upper bound on TDMA-round duration provided by the designer in terms of the maximum number of slots $n_{\max}$ and slot duration $\Delta_{\text{slot}}$. Typically, the choice of $n_{\max}$ depends on the memory available within the communication co-processor such as the number of transmit and receive buffers. Each transmission slot $slot(i)$ within the round has duration $\Delta_{\text{slot}} = \min_i\{size(m_i)\} \times b_j^{\text{speed}}$ µs. The message period $delay(m_i)$, originally expressed in time units, 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 use $delay(m_i)$ to denote this discrete quantity from here on.

The clustering procedure in Fig. 6 takes as input messages $\{m_i\}$ sorted in terms of increasing $period(m_i)$ and returns a set of message clusters where each $C_j$ is allocated to a separate communication bus. Given a set of clusters $\{C_j\}$ and a $k$-FT message $m_i$, we first obtain all feasible message to cluster allocations by grouping $m_i$ with $C_j$ and generating $round(C_j \cup m_i)$. New clusters are created if needed to accommodate all copies of $m_i$. Also, if for the $k$-FT message $m_i$, $n$ feasible message-cluster allocations are obtained, where $n > k$, then the $k$ best solutions are chosen based on bandwidth-utiliza-

```
Procedure CLUSTER(S_msg) /* S_msg := Messages {m_i} sorted by increasing period */
    S_clust := ∅;                    /* Initialize set of message clusters */
    while (S_msg ≠ ∅) begin
        m_i := k-FT message in S_msg with minimum period;
        S_cand := ∅;                 /* 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 round(C_j)) S_cand := S_cand ∪ 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 ∪ S_cand;
            Allocate m_i to (k – n_cand) new clusters and add them to S_clust;
        end;
        if (n_cand ≥ 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;
```

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

tion efficiency–the exact evaluation criterion is discussed later this chapter. The computational complexity of the clustering procedure is $O(n^3)$ where $n$ is the number of messages; the outer **while** loop iterates through all $n$ messages, and during each iteration, ALLOC explores all message to cluster allocations, a process of complexity $O(n^2)$.

**Transmission-Slot Allocation.** Given a message cluster $C_j$ and $m_i$, the ALLOC procedure generates a feasible TDMA round for the new allocation $C_j \cup m_i$. Allocation of messages to multiple buses is related to *bin-packing* where fixed-size objects (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 [17].

An important requirement during slot allocation for the messages in cluster $C_j$ is jitter-free communication. Unpredictable delay or jitter during transmission may lead to missed message deadlines. Figure 7(a) shows multiple TDMA rounds corresponding to messages $m_1$ and $m_2$ with periods $delay(m_1) = 2$ and $delay(m_2) = 5$, 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. Though this allocation satisfies the periodicity requirements of $m_1$ and $m_2$, it results in timing jitter–the minimum and maximum distances between two successive slots for $m_2$ are 4 and 6 slots, respectively. Clearly, this results in a timing violation. Therefore, a minimum-distance constraint between two successive transmission slots for $m_2$ must also be satisfied during allocation. Fortunately, jitter-free transmission can be achieved by appropriately modifying the message periods; Fig. 7(b) shows a jitter-free slot allocation for both messages when $m_2$'s period is modified to 4 slots.

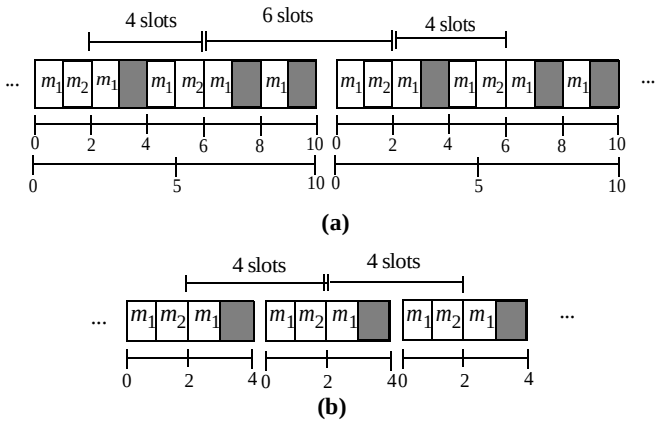



**Fig. 7. (a) A clustering of multiple messages resulting in jitter, and (b) jitter-free slot allocation by appropriately modifying message periods**

The above discussion suggests that the original message periods need modification prior to allocating slots within the TDMA round. Clearly, message periods may be modified in a variety of ways; we adopt a strategy where the periods of all messages within a cluster $C_j$ are constrained to be harmonic multiples of each other. Two messages $m_i$ and $m_j$ have harmonically-related periods if $period(m_i) = 2^k \times period(m_j)$. A similar concept is used in task scheduling in multi-processors where tasks having harmonic periods are allocated to the same processor to increase utilization and minimize completion-time jitter [18] [19]. In [20], we formally prove that allocating messages with harmonically-related periods in FF fashion within $C_j$ guarantees jitter-free transmission. It also maximizes bus utilization and results in a shorter TDMA-round duration thereby reducing the memory requirements of the communication co-processor. Let $p_{\min} = \min_i\{period(m_i)\}$ denote the smallest period among cluster $C_j$'s messages. Then, when allocating a new message $m_i$ to $C_j$, we select its period to be the maximum integer $period(m_i) \le n_{\max}$ satisfying $2^k \cdot p_{\min} \le delay(m_i) < 2^{k+1} \cdot p_{\min}$.

Figure 8 shows the ALLOC procedure which 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 $p_{\max} = \max_i\{period(m_i)\}$. To allocate transmission slots for 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 using the FF packing strategy. Jitter-free transmission of each message $m_i$ is guaranteed if the allotted transmission slots occur in the same positions within each interval $I_k$. Again, the interested reader is referred to [20] where we formally prove that ALLOC generates a communication schedule guaranteeing jitter-free message transmission.

```
Procedure ALLOC (C_j, m_i) /* C_j := Message cluster; m_i := Message */
    S_msg := Set of messages {C_j ∪ m_i} sorted in increasing period order;
    Create an empty TDMA round round(s) with p_max = max_i{period(m_i)} slots;
    while (S_msg ≠ ∅) begin
        m_i := Message with shortest period in S_msg;
        k = p_max/period(m_i) ;        /* k := Number of intervals */
        Divide round(S_msg) into k intervals {I_k}, each of duration period(m_i);
        n := ⌈size(m_i)/Δ_slot⌉ ;    /* Number of slots needed to accommodate m_i */
        for (each interval I_k) begin
            if (n free slots are unavailable) return ∅; /* Allocation is infeasible */
            Allocate n slots within I_k to message m_i in first-fit (FF) fashion;
        end;
    end;
    return round(S_msg); /* Return the feasible allocation */
```

**Fig. 8. The transmission-slot allocation procedure**

**Transmission-Slot Reuse.** 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 the transmission slots allotted to processor $P_k$ among multiple messages. Let $\{m_i\}$ be the set of messages sent by the processor within the message cluster $C_j$. Now, assume message $m_{i+1}$, also transmitted by $P_k$, to be allotted slots within $round(C_j)$. Each message $m_i$ is allotted a number of transmission slots $n_i$ within the time interval $period(m_{i+1})$ in $round(C_j)$. If $n_{reuse}$ denotes the number of slots available for reuse by $m_{i+1}$ with the time interval $period(m_{i+1})$, then

$$n_{reuse} = \sum_i n_i - \sum_i \left\lceil \frac{period(m_i)}{period(T_i)} \right\rceil \times n_i$$

where $period(T_i)$ denotes the period of task $T_i$ transmitting message $m_i$. Therefore, the number of transmission slots to be allotted to message $m_{i+1}$ is $\lceil size(m_{i+1})/\Delta_{slot} \rceil - n_{reuse}$. Given clusters $\{C_j\}$ and the message $m_{i+1}$ 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.

## 5 Case Study

We now illustrate the proposed network construction 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. 9(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-

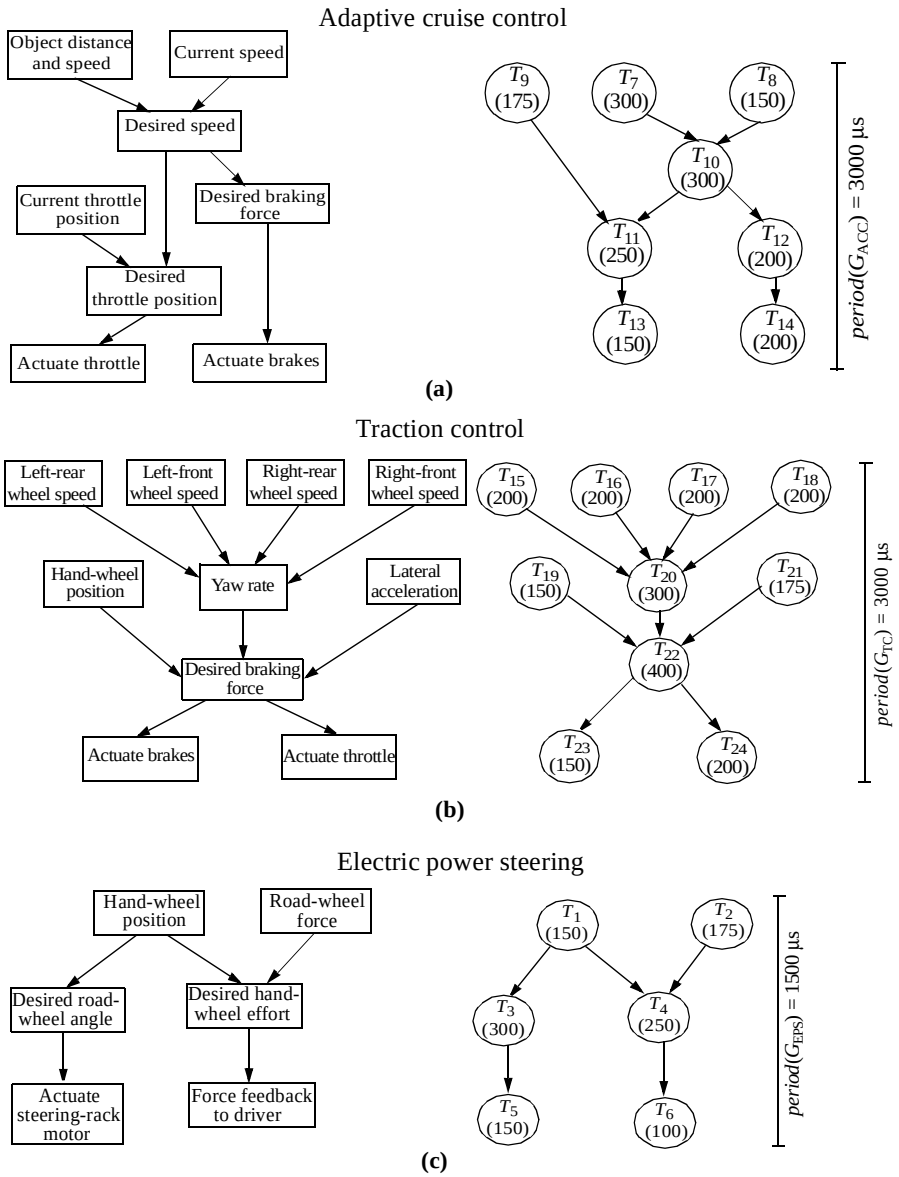
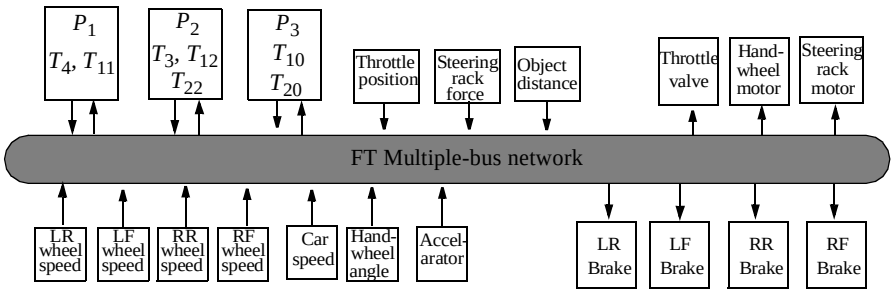



**Fig. 9. The (a) adaptive cruise control, (b) traction control, and (c) electric power steering applications, and the corresponding flow-graph representations**

ure 10(a) shows the physical architecture of the system where sensors and actuators are directly connected to the network and the task-to-processor allocation, while Fig. 10(b) summarizes the various message attributes affecting network topology generation. We assume 1-FT messages throughout. Columns 2 and 3 list the sending and receiving tasks for each message and the message size $size(m_i)$ in bits, respectively, while columns 4 and 5 list the communication delay $delay(m_i)$ for messages in µs, and the trans-




**(a)**

| **Message $m_i$** | **(Sender, receiver)** | ***size*($m_i$) (bits)** | ***delay*($m_i$) (μs)** | ***delay*($m_i$) (slot intervals)** |
|---|---|---|---|---|
| $m_1$ | ($T_1$, $T_3$)<br>($T_1$, $T_4$) | 12 | 300 | 6 |
| $m_2$ | ($T_2$, $T_4$) | 12 | 275 | 5 |
| $m_3$ | ($T_3$, $T_5$) | 20 | 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 |
| $m_8$ | ($T_{10}$, $T_{11}$)<br>($T_{10}$, $T_{12}$) | 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 |
| $m_{16}$ | ($T_{20}$, $T_{22}$) | 22 | 275 | 5 |
| $m_{17}$ | ($T_{21}$, $T_{22}$) | 20 | 1025 | 20 |
| $m_{18}$ | ($T_{22}$, $T_{23}$)<br>($T_{22}$, $T_{24}$) | 12 | 650 | 6 |

**(b)**

**Fig. 10. (a) The physical architecture including task-to-processor allocation, and (b) the message attributes required for network generation**

mission-slot intervals. These delay values are obtained by first assigning deadlines to tasks and then performing a schedulability analysis on their respective processors–a topic discussed previously in Section 3.

We assume a version of the FlexRay communication protocol having a bandwidth of 250 kb/s and a minimum width of 50 μs for the transmission slots in a TDMA round.

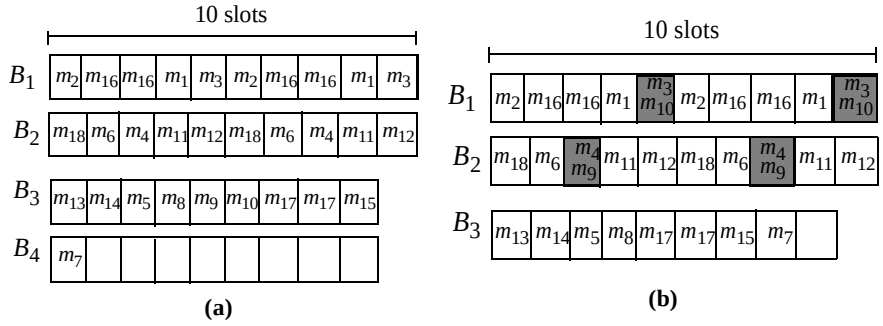



**Fig. 11. Communication schedules generated by ALLOC (a) without slot reuse, and (b) with reuse where the shared transmission slots are shaded**

Figure 11(a) shows the communication schedules generated on buses $B_1$, $B_2$, $B_3$, and $B_4$ without reusing transmission slots. We now show how to share transmission slots between appropriate messages and reduce the number of buses. Consider messages $m_3$ and $m_{10}$ sent by tasks $T_3$ and $T_{12}$, respectively, where both tasks are allocated to processor $P_2$. Message $m_3$'s period is set to its transmission deadline of 300 µs (5 slots) when constructing the TDMA round. Note, however, that the EPS application comprising task $T_3$ has a 1500 µs period; this also corresponds to the time interval between successive $m_3$ transmissions. Therefore, in Fig. 11(a), $m_3$'s transmission needs only one of two allocated slots on bus $B_1$. (Task $T_3$, however, may request $m_3$'s transmission anytime during a TDMA round). Message $m_{10}$ with a period of 10 slots can use the remaining slot. Figure 11(b) shows the schedules obtained by ALLOC with slot reuse; the shared slots between messages $\{m_4, m_9\}$ and $\{m_3, m_{10}\}$, transmitted by processors $P_1$ and $P_2$, respectively, are shaded. Also, slot reuse eliminates the bus $B_4$ in Fig. 11(a).

When TDMA slots are shared between messages sent by a processor, as in Fig. 11(b), 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, an earliest-deadline first approach seems appropriate.

## 6 Conclusion

This paper has addressed the design of low-cost TDMA communication networks for distributed embedded applications. We have developed a fault-tolerant clustering method which allocates and schedules *k*-FT messages on the minimum number of buses to provide jitter-free and predictable transmission. Finally, a case study involving some advanced automotive control applications was discussed and it was shown that the efficient use of communication bandwidth by sharing transmission slots among multiple messages can reduce network topology cost.

This paper does not address the design and implementation of the message scheduler on the co-processors. The message scheduler is responsible for transmitting and receiving messages in their respective slots. We also do not address the fault-tolerant allocation of tasks to processors. The message allocation scheme can be easily incorporated as a subfunction into an overall scheme that deals with both the problems. The above issues will be investigated as part of future work.

## References


[1] E. A. Bretz, "By-Wire Cars Turn the Corner," *IEEE Spectrum*, vol. 38, no. 4, pp. 68-73, April 2001.

[2] G. Leen and D. Heffernan, "Expanding Automotive Electronic Systems," *IEEE Computer*, vol. 35, no. 1, pp. 88-93, Jan. 2002.

[3] R. B. Ortega and G. Borriello, "Communication Synthesis for Distributed Embedded Systems," *Proc. Intl. Conf. Computer-Aided Design* (*ICCAD*), pp. 437-444, 1998.

[4] T. F. Abdelzaher and K. G. Shin, "Combined Task and Message Scheduling in Distributed Real-Time Systems," *IEEE Trans. Parallel & Distributed Syst.*, vol. 10, no. 11, pp. 1179-1191, Nov. 1999.

[5] A. Doboli, P. Eles, Z. Peng, and P. Pop, "Scheduling with Bus Access Optimization for Distributed Embedded Systems," *IEEE Trans. VLSI Syst*., vol. 8, no. 5, pp. 472-491, Oct. 2000.

[6] T-Y. Yen and W. Wolf, "Communication Synthesis for Distributed Embedded Systems," *Proc. Intl. Conf. Computer-Aided Design* (*ICCAD*), pp. 288-294, 1995.

[7] D. L. Rhodes and W. Wolf, "Co-Synthesis of Heterogeneous Multiprocessor Systems using Arbitrated Communication," *Proc. Intl. Conf. Computer-Aided Design* (*ICCAD*), pp. 339-342, 1999.

[8] S. Prakash and A. C. Parker, "Synthesis of Application-Specific Multiprocessor Architectures," *Proc. ACM/IEEE Design Automation Conference*, pp. 8-13, 1991.

[9] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, Boston, 1997.

[10] H. Kopetz, "TTP - A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems," *Proc. IEEE Fault-Tolerant Computing Symp.*, pp. 524-533, 1993.

[11] J. Berwanger et al., "FlexRay - The Communication System for Advanced Automotive Control Systems," *Proc. SAE World Congress*, Paper: 2001-01-0676, 2001.

[12] W. H. Wolf, "An Architectural Co-Synthesis Algorithm for Distributed, Embedded Computing Systems," *IEEE Trans. VLSI Systems*, vol. 5, no. 2, pp. 218-229, Jun. 1997.

[13] B. Kao and H. Garcia-Molina, "Deadline Assignment in a Distributed Soft Real-Time System," *IEEE Trans. Parallel and Distributed Syst*., vol. 8, no. 12, pp. 1268-1274, Dec. 1997.

[14] 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.

[15] C. L. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM*, vol. 24, pp. 46-61, 1973.

[16] X. Hu, J. G. D'Ambrosio, B. T. Murray, and D-L. Tang, "Codesign of Architectures for Automotive Powertrain Modules," *IEEE Micro*, vol. 14, no. 4, pp. 17-25, Aug. 1994.

[17] D. S. Johnson, "Fast Algorithms for Bin Packing," *J. Computer & System Sciences*, vol. 3, no. 3, pp. 272-314, 1974.

[18] K-J. Lin and A. Herkert, "Jitter Control in Time-Triggered Systems," *Proc. Hawaii Intl. Conf. System Sciences*, pp. 451-459, 1996.

[19] C-C. Han, K-J. Lin, and C-J. Hou, "Distance-Constrained Scheduling and its Applications to Real-Time Systems," *IEEE Trans. Computers*, vol. 45, no. 7, pp. 814-826, July 1996.

[20] N. Kandasamy, *Design of Low-Cost Dependable Systems for Distributed Embedded Applications*, Ph.D. Thesis, University of Michigan, 2003.