自動車制御分散システムの静的スケジューリング手法

村 上 尚 彦† 冨 山 宏 之† 高 田 広 章†

現在，自動車の制御系ネットワークでは，従来の CAN（Control Area Network）に代わる，次世代向けの規格として TTP，FlexRay などが提案され，CAN と併用されつつある．TTP と CAN を統合した分散システムのスケジューリングに関する研究は従来より行われているが，そこで用いられているモデルは現実的ではなく，探索の範囲もその手法の性質によって限定されている．本論文では，それらの問題を指摘し，より実用的で柔軟なスケジューリング手法を提案する．我々はその対象として，FlexRay と CAN を統合した自動車制御分散システムを扱う．

A Static Scheduling Method for Distributed Automotive Control Systems

Naohiko Murakami,† Hiroyuki Tomiyama† and Hiroaki Takada†

Today, TTP and FlexRay are developed and expected to be adopted as the next generation automotive control network, together with current CAN (Control Area Network). Scheduling approaches for TTP and CAN based distributed systems have been studied, but they use unrealistic models and the range of the search is limited by the character of their methods. This paper points out such problems and proposes a more practical and flexible method. We target the distributed automotive contorol systems which integrate FlexRay and CAN.

1. はじめに

近年，自動車制御システムでは，省エネルギー，低排気ガス，安全性・利便性の向上などの様々な要求の実現にともない，ネットワーク化が本格化しており，それぞれの目的に特化した様々な規格が考案され，実用化されている[1]．その中でも，特に高いリアルタイム性が要求される制御系では，現在，CAN（Control Area Network）[2] が事実的な標準として広く使われている．しかし，将来的な機能の増加にともなうデータ量の増大，高信頼性の要求，トポロジの複雑化などの問題から，CAN に代わる次世代向けの規格として TTP，FlexRay などが考案され，現在，注目を集めている[1]．

ハードリアルタイムスケジューリング理論の分野においては，システムの各プロセッサ上で動作するタスクや通信のリアルタイム性を検証する手法が多く提案されており，CAN メッセージの最悪遅れ時間解析に関する研究も従来より広く行われてきた[3),4]．しかし，システム全体のリアルタイム性を包括的に検証する手法や，それを用いたスケジューリングに関する研究は少ない．

そのような中で，TTP を用いた分散システムのスケジューリングに関する研究が，Pop らによって進められている[5),6]．また，実際のシステムでは，従来の CAN と，次世代向けの規格を共存させて用いることが想定されており，Pop らも，従来の CAN の最悪遅れ時間解析手法を導入することによって，CAN と TTP に分散されたアプリケーションにも対応したスケジューリング手法を提案している[7]．

そこでは，スケジューリングの手順を決定項目ごとに大きく，(i) タスクの実行ノードの割付け（マッピングとよぶ），(ii) 送信するメッセージの組合せの決定（フレームパッキングとよぶ），(iii) タスクの起動時刻とメッセージの送信時刻の決定（タイムスケジューリングとよぶ）の 3 つの段階に分けている．そして，**図 1** の (a) に示すように，各段階でそれぞれの決定項目に対する探索候補を順番に選択し，最終的に得られるスケジューリング結果に対して時間制約を満たしているかどうかを評価する．時間制約を満たした場合にはその結果を出力し，そうでない場合には，新たな探

† 名古屋大学大学院情報科学研究科情報システム学専攻
Department of Information Engineering, Graduate School of Information Science, Nagoya University

索候補を選択し，再度その結果に対して評価を行う．すべての探索候補について評価を終えるまでこの操作を繰り返し，いずれも時間制約を満たせなかった場合には，「スケジューリング不能」として終了する．

この方法では，一般的に，後の段階ですべての探索候補を評価し終えるまで前の段階における決定項目が変更されないため，探索に偏りが生じ，対象の性質によっては多くの時間を必要とする．Pop らの手法では，システム全体の動作周期が統一されていることを前提としており，それに基づいて (iii) では近似的に最適な候補を一意に選択することによって，この問題を緩和している．しかし，複数の異なったアプリケーションが複雑に関連して動作する実際のシステムでは，そのようなモデルは現実的ではないため，この方法を適用することはできない．

本研究では，現在，標準化が世界的に進められている FlexRay[8] と CAN を統合して実現される，将来的なシステムを対象としたスケジューリング手法を提案する．本手法では，図 1 の (b) に示すように，従来手法で個別に扱っていた決定項目を統一して 1 つの探索候補を選択することで，偏りのない効率的な探索を実現する．またそこでは，従来のアプリケーションモデルでの周期に対する前提が，実際のシステムを表現する際には不十分であることを指摘し，それを現実的な形に拡張したモデルを扱う．さらに，従来の手法では考慮されていなかった時間制約以外の要件として，システムの負荷をスケジューリング結果の評価に加えることによって，より実用的な手法の実現を目指す．

本論文では，2 章で対象とするシステムの構成や，CAN と FlexRay の概要について述べた後，本手法で使用するアプリケーションモデルを従来のモデルと比較して説明する．3 章では，本手法で決定する項目について整理した後，その指標となる制約の表現方法について述べる．新たに考慮した要件や，探索範囲の拡張部分についても，ここで説明する．4 章では，従来の探索手法の問題点について指摘しつつ，本手法の具体的なアルゴリズムについて説明する．5 章では，本手法を適用した実験を通してその有効性を示す．

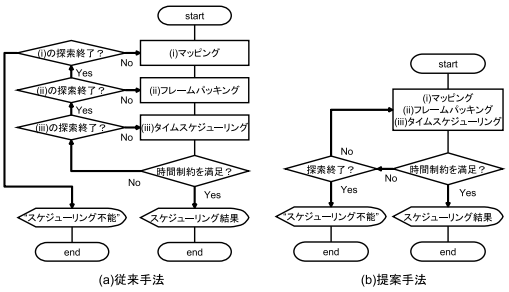


図 1 スケジューリング手順の比較
Fig. 1 Comparison of the scheduling methods.

2. システム構成

これまで，自動車制御システムでは様々な要求の実現にともない，従来の機械的な制御から電子制御への移行が段階的に進められてきた．そして現在では，FlexRay などの次世代向けのネットワーク規格の誕生により，ブレーキやステアリングなどの非常に高い信頼性が求められる制御においても，X-By-Wire と呼ばれる電子制御システムへの移行が実現されようとしている[8]．

しかし，現在 CAN で実現されているシステムが，すべて FlexRay などの次世代の規格に置き換えられるのではなく，コストの削減やリソース使用率の向上などのため，それらの規格は共存して用いられることが想定される[1]．その結果，物理的な制約や，将来的な拡張により，同一のアプリケーションに属するタスクが，異なった複数の規格のネットワーク上に分散されて配置されることがある．そのため，異なった規格間の通信を可能にするゲートウェイを用いて，それぞれのサブネットワーク間を接続したシステムの構成が現実的となっている．

本論文では，**図 2** のような，CAN と FlexRay のそれぞれで構成されたサブネットワークを，1 つのゲートウェイで接続したシステムを対象モデルとして扱う．システム上のノード（ECU）はすべて，センサやアクチュエータとの入出力インタフェース，ホスト CPU，それぞれのネットワーク用の通信コントローラ，および，ホスト CPU-通信コントローラ間インタフェースで構成される．また，ゲートウェイは，CAN，FlexRay のそれぞれの通信コントローラを持ち，サブネットワーク間で受け渡されるメッセージの中継を行う．

本章では，CAN，FlexRay のそれぞれの通信方法，および，本手法で扱うアプリケーションモデルについて説明する．

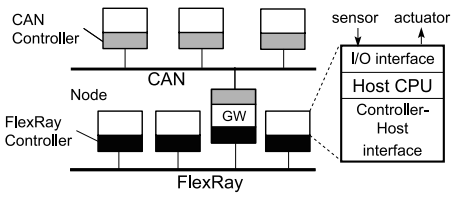


図 2 システム構成
Fig. 2 System architecture.

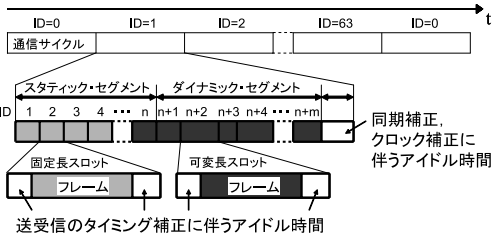


図 3 FlexRay 通信サイクル
Fig. 3 FlexRay communication cycle.

2.1 CAN

CAN は最高で 1 Mbps の転送速度を実現できるブロードキャストバスである[2]．CAN バス上では，最大長を持つフレーム単位でデータがやりとりされ，1 つのフレームあたり最大 8 バイトのデータを通信できる．それぞれのフレームには 11 ビットの固有の ID が割り付けられる．ID はメッセージの送信時の優先度を表し，受信時のフィルタリングにも用いられる．

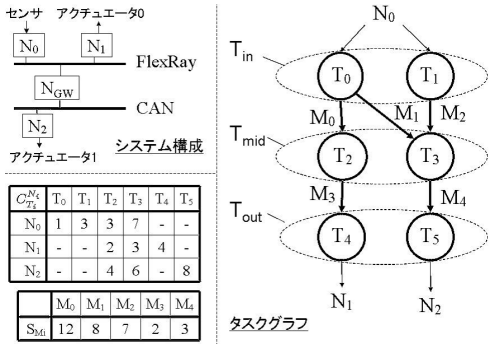
CAN では，すべてのノードがそれぞれの送信要求タイミングでフレームの送信を開始する．バス上にメッセージが流れていないときには先にバスにアクセスしたノードが送信権を獲得する．衝突した場合には，最も小さい ID を持つフレームが送信され，他のノードはその送信終了まで待ち，再び送信を試みる．詳細については文献 2) を参照されたい．

2.2 FlexRay

FlexRay は CAN と同様ブロードキャスト型のネットワークだが，通信速度は最大 10 Mbps と高速である．また，トポロジもバス型に加え，スター型，および，それらを混在させたハイブリッド型などにも対応し，柔軟性を向上させている[8]．

FlexRay では，あらかじめ衝突が起こらないように，各ノードに静的に送信時間が割り当てられる．**図 3** に示すように，FlexRay の通信は 64 回の通信サイクルを基準に行われる．各通信サイクルには順番に 0〜63 までのサイクル ID が割り振られ，その通信時間はスタティック・セグメントとダイナミック・セグメントに分けられる．各セグメントは，ノードの送信単位である複数のスロットに分割される．各スロットは，通信サイクルの先頭から順に，スロット ID が 1 からそのスロット数だけ割り振られ，最大 254 バイトのデータを含むフレームが送信される．すべての通信サイクルにおいてスロットの数は同じであり，各サイクルの同じ ID のスロットでは同一のノードが送信を行う．ただし，送信されるフレームに関しては，通信サイクルごとに異なっていてもよい．

スタティック・セグメントでは，各スロットがつねに同じ長さに固定されており，あらかじめ静的に決められたスケジュールによって，確実に通信が保証される．一方，ダイナミック・セグメントでは，送信されるフレームの大きさに合わせてスロットの長さが変動するため，後方のスロットに割り当てられたフレームが，先方のフレームに邪魔されて送信できない場合がある．詳細については，文献 8) を参照されたい．



$C_{T_i}^{N_i}$	T_0	T_1	T_2	T_3	T_4	T_5
N_0	1	3	3	7	-	-
N_1	-	-	2	3	4	-
N_2	-	-	4	6	-	8

	M_0	M_1	M_2	M_3	M_4
S_{M_i}	12	8	7	2	3

図 4 タスクグラフの例
Fig. 4 An example of task graph.

各セグメントの割合は，システムの設計者がその方針に従って決定することができる．本論文では，システムのリアルタイム制約を確実に保証することを目的としているため，通信サイクルはスタティック・セグメントのみで構成されるものとする．なお，従来手法[7] で対象としている TTP では，動的に通信時間を変更することはないため，この仮定によって FlexRay は TTP に近い通信方式をとることになる．しかし，TTP では，1 回の通信サイクルにおいて各ノードには 1 つしかスロットを割り当てられず，また，各スロットの長さはノードによって異なっているため，その送信スケジュールは FlexRay と大きく異なる．これは，従来手法を本手法の対象としているシステムに対してそのまま適用することができないことを意味する．

2.3 アプリケーションモデル

本論文では，対象とするアプリケーションを**図 4** に示すような，タスクグラフとよばれる有向グラフを用いて表現する．タスクグラフの頂点（図の丸）と辺（図の矢印）は，処理単位であるタスクと，タスク間のデータの流れを示すメッセージをそれぞれ示している．N_i は FlexRay または CAN ネットワーク上のノードを表しており，センサやアクチュエータに対して直接処理を行うタスクには，それらが接続されたノードとの入出力関係をグラフ中に示している．両ネットワークの中継を行うゲートウェイノードは，N_{GW} で表す．

タスク T_i はノード N_i での最大実行時間 $C_{T_i}^{N_i}$ と周期 P_{T_i}，メッセージ M_i はデータサイズ S_{M_i} と

周期 P_{M_i} をそれぞれ持つ．M_i の送信元のタスクを $T(M_i)$，受信先のタスクを $R(M_i)$ で表すこととする．タスクは，センサからの入力処理を行う入力タスク T_{in}，アクチュエータへの出力処理を行う出力タスク T_{out}，および，それらの間で計算と中継を行う中間タスク T_{mid} のいずれかに属する．図 4 では，$T_{in} = \{T_0, T_1\}$，$T_{mid} = \{T_2, T_3\}$，$T_{out} = \{T_4, T_5\}$ となる．

$C_{T_i}^{N_i}$ は，ノード N_i でのタスク T_i の 1 回の実行に要する最大の時間を表し，あらかじめ設計段階で計測的あるいは解析的に与えられるものとする．T_{in}，T_{out} に属するタスクは，物理的な制約により，あらかじめそれらが処理すべきセンサやアクチュエータを持つノードに決定されているものとするため，最大実行時間もそれらが処理されるノードに対してのみ与えられる．

$C_{T_i}^{N_i}$ には，T_i が送信，受信するすべてのメッセージデータを，N_i のホスト CPU と通信コントローラの間で転送する時間が含まれる．そのため，FlexRay，CAN のいずれにおいても，タスクの終了時刻には，そのタスクが送信するメッセージをただちに送信でき，同様に，メッセージの送信完了時刻には，そのメッセージを利用するタスクがただちにそれを利用して実行することができるものとする．なお，同じノードで処理されるタスク間のメッセージは，ノード内のメモリを介して受け渡されるため，送信元のタスクの終了時刻には，受信先のタスクがそのメッセージをただちに利用して実行することができる．このようなメッセージを「ノード内メッセージ」とよぶこととする．

P_{T_i} はタスクの起動間隔，P_{M_i} はメッセージの送信要求の発生間隔をそれぞれ表す．従来の手法[7] で用いられているモデルでは，グラフ全体で周期がすべて同じ値に統一されており，また，それが設計段階に要求される仕様として与えられることが前提となっている．

通常，タスクとその送受信メッセージに異なった周期を設定した場合，処理や通信に無駄が生じる．しかし，異なった周期を持つ複数のタスクが，1 つの同じタスクから得られるデータを用いる場合，部分的にそれらの周期を異なった値に設定することも考えられる．たとえば，図 4 で，アクチュエータ 0 の制御を周期 4，アクチュエータ 1 の制御を周期 8 で行うため，T_0，T_2，T_4 を周期 4，T_1，T_3，T_5 を周期 8 に設定したとする．このとき，T_0 から生成されるメッセージ M_0，M_1 は，T_2，T_3 の周期に合わせてそれぞれの周期が 4，8 に設定されるのが自然である．このように，1 つのアプリケーションを同一の周期のタスクとメッセージで表現することは，実際のシステムでは現実的ではない．

そこで，本手法では，タスクグラフ中の各タスク，メッセージに，それぞれ独立した周期を設定できるものとした．また，次章で述べるように，それらを本手法の決定項目として探索の対象に含める．ただし，それらがとりうる値は，(FlexRay の通信サイクル長) $\times\ 2^n (n = 0, 1, 2, \cdots)$ に制限する．これは，FlexRay で各メッセージを受信するときに，通信コントローラがそのサイクル ID とスロット ID を用いてフィルタリングを行うためである．最大サイクル ID は 64（$= 2^6$）と決まっているので，メッセージの周期をそれ以外の値に設定すると，フィルタリングをソフトウェアで別に行う必要があり，その負荷が大きくなってしまう．

3. 問題定義

本章では，本論文で扱う問題について定義する．はじめに，本手法で決定する項目について述べ，次に，その決定に必要となる制約や要件について述べる．

3.1 決定項目

本手法で決定する項目は，主に以下の (a)〜(i) に分類できる．本手法でのスケジューリングとは，本章の以降の節で述べる制約や要件に応じて，これらのすべての項目を静的に（システムの実行前に）決定することをいい，1 つ 1 つの異なった決定結果をスケジュールとよぶ．

(a) タスクとメッセージの周期
(b) 中間タスクの処理ノード
(c) FlexRay スロットの送信ノード
(d) FlexRay メッセージのフレームパッキング
(e) FlexRay フレームの送信スロット
(f) CAN メッセージのフレームパッキング
(g) CAN フレームの優先度
(h) FlexRay タスクの起動時刻
(i) CAN タスクの優先度

図 5 は，図 4 のモデルに対するスケジュールの例を示している．ここでは，FlexRay の通信サイクルの長さを 4，各スロットの長さを 2（1 通信サイクル中に 2 つのスロット），フレームのデータサイズを FlexRay では 16，CAN では 8 とした．図にも表れているように，FlexRay と CAN では，それぞれの通信方式に応じて異なった決定項目を持つ．

FlexRay では，スタティック・セグメントのみを用いるという前提のもとで，すべてのメッセージについてあらかじめ静的にその送信時間が割り当てられるた

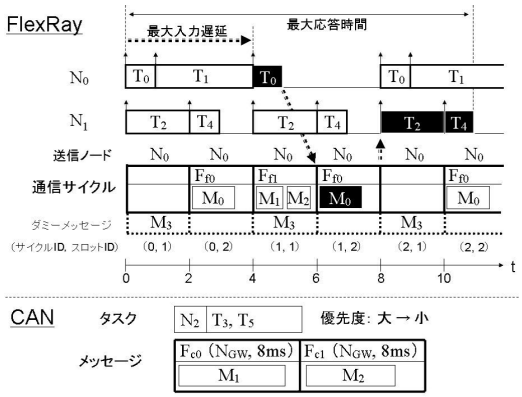


図 5 図 4 に対するスケジュールの例
Fig. 5 An example of the schedule for Fig. 4.

め，タスクに関しても，メッセージと同様，静的に実行時間を割り当てることによって，処理全体の動作を設計時に完全に決定・予測できる．一方，CAN では，メッセージは優先度に従って動的にその送信時間が決まるため，タスクについても，その優先度だけを決定項目とし，実際の実行時間はその優先度に基づいて動的に決定されるものとする．

(a) では，タスクグラフ中の各タスク，メッセージの周期を決定する．2.3 節で述べたように，従来の手法では，これらの値がすべて設計段階で要求される仕様として与えられていた．しかし，タスクやメッセージの周期もその他の項目と相互に関連し，システムの応答性やその他の要件に影響を与えるため，それらを独立して決定することは本質的ではない．そこで本手法では，タスクとメッセージの周期も決定項目に含めることとする．図 5 では，T_0，T_2，T_4，M_0，M_3 が周期 4，T_1，T_3，T_5，M_2，M_4 が周期 8 となっている．

(b) では，前章で述べたアプリケーションモデルにおける，中間タスク T_{mid} に属する各タスクを処理するノードを決定する．図 5 では，T_2 が N_1，T_3 が N_2 にそれぞれ割り付けられている．

(c) では，FlexRay の各スロット ID に対する送信ノードを決定する．図 5 では，1，2 のいずれのスロット ID に対しても，ノード N_0 が割り当てられている．

(d)，(f) では，FlexRay，CAN のそれぞれで送信されるフレームのメッセージ構成を決定する．1 つのフレームは同じノードから送信されるメッセージのみで構成されなければならない．また，原理上は可能だが，スケジューリングとソフトウェアの複雑化を避けるため，ここではメッセージを異なったフレームに分割してはならないものとする．ただし，FlexRay において，周期の関係上，1 つのメッセージが異なった複数のメッセージと別々のフレームを構成することは許すものとする．CAN については，すべてのフレームを同じ周期のメッセージのみで構成する．

図 5 の FlexRay 上では，フレーム F_{f0} が M_0 のみ，F_{f1} が M_1 と M_2 でそれぞれ構成されている．CAN 上では，フレーム F_{c0} が M_1 のみ，F_{c1} が M_2 のみでそれぞれ構成されている．M_3，M_4 はノード内メッセージであるため，いずれのフレームにも含まれない．

(e) では，FlexRay で送信される各フレームを送信するスロットを決定する．2.3 節で述べた前提より，すべてのメッセージの周期は FlexRay 通信サイクル長の整数倍となるため，各フレームは，その周期ごとの通信サイクルにおいて，同一のスロット ID のスロットにて送信される．したがって，FlexRay フレームの送信スケジュールは，それが最初に送信されるスロットのサイクル ID と，スロット ID によって決定される．このとき，各フレームは，(c) によってそれらが送信されるノードが割り当てられたスロット ID のスロットでのみ送信できる．図 5 では，F_{f0} がサイクル ID $= 0$，スロット ID $= 2$ から，周期 4 で送信されている．F_{f1} は，サイクル ID $= 1$，スロット ID $= 1$ から，周期 8 で送信されている．

(g) では，CAN で送信される各フレームの優先度を決定する．CAN ではこの優先度に基づいて各フレームに ID が割り付けられる．図 5 では，F_{c0}，F_{c1} の順で高い優先度を設定している．

(h) では，FlexRay 上のノードに割り付けられたタスクについて，その起動時刻を決定する．原理上，FlexRay 上のノードでもタスクのプリエンプションは実現可能だが，ここではスケジューリングの複雑化を避けるため，ノンプリエンプションを仮定する．図 5 では，各タスクの起動時刻が上向きの矢印で示されている．

(i) では，CAN 上のノードに割付けられたタスクについて，その優先度を決定する．タスクの優先度は各ノードにおいて独立に与え，それぞれのノード内でのタスクの実行時間はこれらに基づいて動的に決定される．図 5 では，CAN 上のノードは N_2 のみとなっており，そこに割り当てられたタスクには T_3，T_5 の順で高い優先度を設定している．

3.2 制約条件

本手法では，スケジューリングの制約条件として，システムの時間制約と負荷の削減を考慮に入れる．従来の手法ではシステムの負荷には着目していないが，タスクのノード割付けやメッセージのフレームパッキ

ングの方法によっては，プロセッサやネットワークにかかる負荷が大きく変わってくる．また，本手法では周期を探索項目に含めているため，時間制約だけを考慮してスケジューリングを行った場合，周期が必要以上に小さく設定され，システムにかかる負荷が著しく大きくなってしまうことが考えられる．

さらに本手法では，従来の時間制約の表現方法や応答時間に対する計算方法を，周期の探索に合わせてより実用的な形に拡張した．以降では，その具体的な内容について説明する．

従来の手法[7] では，タスクグラフ全体の応答時間に対して 1 つの制約を与えている．しかし，2.3 節でも述べたように，一般的に複数のタスクが同一のタスクから得られるデータを共有することは多く，個々のアプリケーションを 1 つのグラフとして扱い，すべての制約を 1 つにまとめることは容易ではない．

そこで，本手法では，下に定義するタスクグラフ中の任意の「経路」の応答時間に対して制約を与える．これによって，システムの任意の処理の流れに対して制約を表現することが可能になる．ここでは，混同を避けるため，タスクとメッセージを一般化してそれぞれ t_i と m_i で表す．

経路 ：タスク t_i とメッセージ m_i の交互系列 $(t_0, m_0, t_1, m_1, \ldots, t_{k-1}, m_{k-1}, t_k)$，（$k \geq 0$）で，$t_i = T(m_i) \wedge t_{i+1} = R(m_i)$，（$i = 0, \cdots, k-1$）を満たすとき，この交互系列を経路という．また，$t_0$ を経路の始点タスク，t_k を終点タスクとよぶ．

具体的には，指定した経路にデッドラインを定め，その最大応答時間をデッドライン以下の範囲でできるだけ小さくすることを目標とし，そのための指標を与えるが，その詳細については次章で述べる．

たとえば，図 4 における系列 $(T_0, M_0, T_2, M_3, T_4)$ は 1 つの経路であり，その始点タスクは T_0，終点タスクは T_4 となる．はじめに，この経路（P とする）を用いて，FlexRay 上の経路での最大応答時間の計算方法を説明する．

FlexRay 上では，タスクの実行時間とメッセージの送信時間がすべて静的に決まっているので，スケジュール上を単純にたどっていけば，その応答時間を求めることができる．たとえば，図 5 において，P の T_0〜T_4 までの最大応答時間を求めてみると，図中の黒塗りの部分をたどった時間，つまり，$11 - 4 = 7$ となる．ここで，注意しなければならないことが 2 点ある．

1 つは，本手法では，センサによって入力される値が実際に用いられるまでの時間（これを入力遅延とよぶ）も経路の応答時間に含めるということである．入力遅延は，始点タスクの起動直後にセンサによって新しい値が入力されたときが最悪となり，その値は始点タスクの周期で与えられる．つまり，P の最大応答時間は，図 5 に示すように，先ほど求めた時間に対して P_{T_0} を加え，$4 + 7 = 11$ となる．

入力遅延は，周期を決定項目に入れるうえで重要になる．たとえば，P の応答時間に対して，入力遅延を考慮せず，そのデッドラインを 7 にしたとする．このとき，図 5 ではその制約を満たしていることになる．ここで，このスケジュールを維持したまま，経路内のすべての周期を倍の値に変更したとする（つまり，黒塗りの部分が周期 8 で繰り返される）．すると，その場合でも T_0〜T_4 までの応答時間は変わらないため，依然として制約を満たす．これは，入力遅延を制約に考慮しなかった場合，周期は無限に大きくすることができることを意味する．いい換えると，入力遅延を制約に考慮することによって，現実的に必要な周期を決定することができることを示している．

2 つ目は，経路の応答時間は必ずしも一定でないということである．P の場合，経路に含まれるすべてのタスクとメッセージが同じ周期となっている．このとき，経路の応答時間はつねに一定の値となる．しかし，経路に複数の異なった周期を持つタスクとメッセージが存在する場合は，その応答時間は複数のパターンに分かれるため，それらの中で最大となるパターンを選択しなければならない．たとえば，図 5 で T_0 と M_1 の間の遅延時間を考える．$t = 0$ に起動される T_0 に対しては，$t = 4$ のスロットで M_1 を送信できるため，その遅延時間は $4 - 1 = 3$ となる．一方，$t = 4$ に起動される T_0 に対しては，$t = 12$ のスロットまで M_1 を送信できないため，その遅延時間は $12 - 5 = 7$ となる．一般的に，FlexRay 上の経路の応答時間のパターン数は，(経路内のタスクまたはメッセージが持つ最大の周期)/(始点タスクの周期) で与えられる．

次に，CAN 上での最大応答時間の求め方を説明する．CAN では，タスクの応答時間，メッセージの送信時間がともに，優先度に従って動的に決定される．そのため，FlexRay のように単純にスケジュールをたどることはできない．リアルタイムスケジューリング理論の分野では，静的優先度ベース，プリエンプティブなタスクに対するスケジューリング手法[9] が確立されており，本手法でもそれを用いてタスクの最大応答時間を決定する．また，メッセージについても，冒頭で述べた CAN の最悪遅れ時間の解析手法[3] を用いて，その最大送信時間を計算することができる．

CAN 上でのタスクとメッセージの間の遅延時間は，

上で述べた入力遅延の考え方と同様にして与えることができる．すなわち，あるメッセージ M_i について，その送信元タスク $T(M_i)$ と M_i の間の最大遅延時間は，P_{M_i} となり，M_i とその受信先タスク $R(M_i)$ の間の最大遅延時間は，$P_{R(M_i)}$ となる．

最後に，GW を通過するメッセージの，GW 滞留時間を考える．これは，送信元のネットワークでの送信が完了し，GW にメッセージが到着してから，受信先のネットワークでの送信要求が発生するまでの待ち時間と考えることができる．この最大時間も入力遅延の考え方と同様にして与えることができ，あるメッセージ M_i について，その最大 GW 滞留時間は，FlexRay から CAN，CAN から FlexRay のいずれの方向に送信される場合においても，その周期 P_{M_i} となる．

4. アルゴリズム

本章では，本手法のアルゴリズムについて説明する．はじめに，その基本となるシミュレーティド・アニーリング（SA）について説明した後，そこでの決定項目と制約や要件の表現方法について述べる．

4.1 シミュレーティド・アニーリング（SA）

一般的に，スケジューリングの探索候補の数はシステムの規模に対して指数的に増大する．1 章でも述べたように，従来手法[7] では，図 1 の (a) に示した (iii) タイムスケジューリングにおいて，近似的に最適な候補を一意に選択することにより，探索時間の削減を実現している．その方法は，タスクグラフの処理順序に従って，先頭からタスクとメッセージを詰めていく，というシンプルなものである．しかし本手法では，2.3 節で述べたように，グラフ内で周期が統一されておらず，処理の実行順序を明確に決めることができないため，この方法を適用することはできない．

本手法では，組合せ最適化アルゴリズムの 1 つであるシミュレーティド・アニーリング[10]（以下，SA とよぶ）を用いることによって，効率的なスケジューリングの探索を目指す．本手法で対象とするシステムは将来的なものであり，必要とされる制約や要件がいまだ明確でないため，モデルの変化に柔軟に対応できることが望ましい．SA は，組合せ最適化アルゴリズムの中でも最も汎用的で容易に拡張・変更が行えることが知られており，本手法に適したものであるといえる．**図 6** にそのアルゴリズムを示す．

SA の探索は，ある初期解から始まり，近傍解と呼ばれる元の解を少し変化させた解へ遷移する作業を繰り返す．解の評価は，あらかじめ定められた方法によって計算されるコストによって表現する．遷移は，元の解と比較して近傍解のコストが小さければ無条件で遷移し，そうでないときは温度に基づいた確率で遷移する．探索の過程で温度を徐々に下げ，この確率を徐々に減らすことで，近似的に最適解を求めていく．その確率や終了条件の与え方は，アルゴリズムの設計者によって様々なものがある．

1. 初期解xを生成、初期温度tを決定．（C(x)：x のコスト）
   暫定最適解best_S := x ．暫定最適コスト best_C := C(x)
2. xから近傍解x'を生成し、終了条件（回数、温度、収束度合いなどにより判定）をみたすまで以下を反復
   - C(x̂)≦C(x)ならx := x̂.
     （かつC(x̂) < best_Cならbest_S := x', best_C := C(x̂). ）
   - C(x̂)>C(x)なら確率$e^{-\Delta/t}$で x := x̂.　（Δ=C(x̂)-C(x)）
3. 温度tを更新（冷却）してステップ2へ

図 6 SA アルゴリズム
Fig. 6 SA algorithm.

タスク

タスク	T_0	T_1	T_2	T_3	T_4	T_5
優先度	3	1	4	2	6	5
周期	4	8	4	8	4	8
処理ノード	N_0	N_0	N_1	N_2	N_1	N_2
割当て基準	M_0	M_1	M_0	M_2	M_3	M_4
オフセット	0	0	0	0	0	0

メッセージ （＊：空メッセージ）

メッセージ	M_0	M_1	M_2	M_3	M_4	＊
周期	4	8	8	4	8	-
FlexRay順序	5	1	3	2	6	4
FlexRayオフセット	0	1	1	0	0	-
CAN順序	1	4	5	2	3	-

図 7 SA における解の例
Fig. 7 An example of the solution in SA.

SA は，解とその近傍，解に対するコストの計算方法を定義できれば，様々な問題に対して適用できる．以降の節では，本手法でのそれらの定義を説明する．

4.2 解の表現

ここでは，本手法の SA での解の表現方法について説明する．例として，**図 7** に，図 5 のスケジュールに対応した解を示す．

解はそれ自体がスケジュールを示すのではなく，スケジュールを導出するためのパラメータの集合となっている．その構成は大きく，タスク，メッセージのそれぞれの割当てに関するパラメータに分かれる．タスクのパラメータには，優先度，周期，処理ノード，割当て基準，オフセットがある．メッセージのパラメータには，周期，FlexRay 順序，FlexRay オフセット，CAN 順序がある．具体的に，図 7 の例を用いて，これらのパラメータの値に基づいて 3.1 節に示した (a)〜(i) の項目を決定する手順を以下に示す．

(1) はじめに，(a) タスクとメッセージの周期は，解のタスクの周期，メッセージの周期によってそれぞれ与えられる．本手法では，あらかじめ周期の範囲を限定することによって，探索空間が無駄に大きくならないようにしている．

(2) (b) 中間タスクの処理ノードは，解のタスクの処理ノードによって与えられる．前提により，入力タスクと出力タスクはその割付けがあらかじめ決められ，固定されている．この結果により，各メッセージを送信，受信するノードも同時に決定される．

(3) 3.1 節では，(c) FlexRay スロットの送信ノード，(d) FlexRay メッセージのフレームパッキング，(e) FlexRay フレームの送信スロットをそれぞれ別々の項目として説明したが，ここでは，解のメッセージの FlexRay 順序と FlexRay オフセットを用いてこれらをまとめて決定する．FlexRay 通信サイクル長を L_{cyc}，通信サイクル中のスロット数（=最大スロット ID）を N_{slot} として，以下でその決定方法を説明する．

解の FlexRay オフセットは，各メッセージが最初に送信されるスロットのサイクル ID を示している．メッセージ M_i の FlexRay オフセットの範囲は，0〜$(P_{M_i}/L_{cyc}-1)$ となる．

解の FlexRay 割当て順序は，各メッセージの送信スロット ID を決定する順序を示している．その決定方法は，空の通信サイクルに順にメッセージを埋めていくような形式をとる．すなわち，FlexRay 割当て順序の値の小さいメッセージから順に，$\mathrm{ID}=1,2,3,\cdots,N_{slot}$ の順で各スロットについて，そのメッセージが送信可能であるかどうかを調べ，はじめに見つかった送信可能なスロット ID を，そのメッセージの送信スロット ID として決定する．そして，送信スロット ID が決定したとき，メッセージを実際に送信されるスロットの中に割り当てる．メッセージ M_i が FlexRay オフセット $=j$，スロット $\mathrm{ID}=k$ で送信可能である条件は，次の (i)，(ii) を同時に満たすことである．

(i) スロット $\mathrm{ID}=k$ に割り付けられた送信ノードと，M_i を送信するノードが一致するか，または，スロット $\mathrm{ID}=k$ にいまだ送信ノードが割り付けられていない．

(ii) スロット $\mathrm{ID}=k$，サイクル $\mathrm{ID}=m$（$m=j$, $j+\frac{P_{M_i}}{L_{cyc}}$, $j+2\frac{P_{M_i}}{L_{cyc}}$,$\cdots$, $j+(\frac{64L_{cyc}}{P_{M_i}}-1)\cdot\frac{P_{M_i}}{L_{cyc}}$）のすべてのスロットが，$S_{M_i}$ 以上の空きサイズを持つ．

空きサイズは，スロットに追加することのできるデータのサイズを示している．初期状態では，すべてのスロット ID に対して送信ノードは割り付けられておらず，1 つのメッセージも割り当てられていない．そのため，すべてのスロットの空きサイズは，あらかじめ与えられる FlexRay のフレームのサイズに一致する．各スロット ID に対する送信ノードは，いずれかの通信サイクルにメッセージが割り当てられた時点で，そのメッセージを送信するノードに確定される．また，そのとき，メッセージが割り当てられたすべての通信サイクルのスロットの空きサイズは，そのメッセージのデータサイズ分だけ減少する．

解の FlexRay 順序の値は，すべてのメッセージに対して与えられるが，FlexRay 上で送信されることのないメッセージについては，この値は意味をなさない．ただし，FlexRay 上のノード内メッセージについては，それをダミーメッセージとしてスロットに擬似的に割り当てる．その際には，上で述べた条件 (i)，(ii) には関係なく，無条件で割り当てられるものとし，その結果として送信ノードや空きサイズはいっさい変更されないものとする．この意味については，後でタスクの起動時刻の決定について説明する際に述べる．

また，解のメッセージの中には，FlexRay 順序の値だけを持つ＊で示された「空メッセージ」が，$(N_{slot}-1)$ の数だけ存在する．これは，スロットの割当て時に，探索を開始する先頭のスロット ID を 1 つだけ増やす働きを持つ．空メッセージを用いることによって，メッセージを完全に先頭から詰めていくだけでなく，より広い範囲でメッセージを配置できるようになる．

図 7 では，はじめに，FlexRay 順序の値が 1 となっている M_1 からスロットの割当てを決定する．M_1 は，FlexRay オフセット $=1$，$P_{M_1}=8$ より，サイクル $\mathrm{ID}=1,3,5,\cdots,63$，スロット $\mathrm{ID}=1$ のスロットに割り当てられる．このとき，スロット $\mathrm{ID}=1$ の送信ノードが N_0 に確定される．また，割り当てられたスロットの空きサイズは $16-8=8$ となる．

次に，M_3 は，ノード内メッセージであるため，ダミーメッセージとしてサイクル $\mathrm{ID}=0,1,2,\cdots,63$，スロット $\mathrm{ID}=1$ のスロットに割り当てられる．

次に，M_2 は，その送信ノードが N_0，FlexRay オフセット $=1$，$S_{M_2}=7$，$P_{M_2}=8$ であるため，M_1 と同じスロットに配置される．割り当てられたスロットの空きサイズは $8-7=1$ となる．このようにして，同じスロットに割り当てられたメッセージによってフレームが構成される．

次に，空メッセージ＊によって，後に続く M_0 はスロット $\mathrm{ID}=2$ から割り当てられるスロットを探索することになり，サイクル $\mathrm{ID}=0,1,2,\cdots,63$，スロット $\mathrm{ID}=2$ のスロットに割り当てられる．その結果，

スロット ID = 2 の送信ノードも N_0 に確定される．

最後に，M_4 は CAN 上のノード内メッセージなので，ここでの割当てには関与しない．

(4) (f) CAN メッセージのフレームパッキング，(g) CAN フレームの優先度は，解のメッセージの CAN 順序に従って決定される．考え方としては，先ほどの FlexRay と同様であり，CAN 順序の値の小さいメッセージから，フレームを割り当てていく．CAN の場合は，優先度の高い順に無限に並ぶフレーム $F_{c0}, F_{c1}, F_{c2}, \cdots$ の中にメッセージを埋めていくような形で，そのフレームパッキングと各フレームの優先度を決定する．この際，メッセージ M_i がフレーム F_{cj} に割り当て可能となる条件は，次の (i)，(ii)，(iii) を同時に満たすことである．

(i) F_{cj} の送信ノードと，M_i を送信するノードが一致するか，または，F_{cj} にいまだ送信ノードが割り付けられていない．

(ii) F_{cj} の周期と，P_{M_i} が一致するか，または，F_{cj} の周期がいまだ決まってない．

(iii) F_{cj} が S_{M_i} 以上の空きサイズを持つ．

空きサイズは，フレームに追加することのできるデータのサイズを示している．初期状態では，すべてのフレームに対して送信ノードと周期が決まっておらず，1 つのメッセージも割り当てられていない．そのため，すべてのフレームの空きサイズは，あらかじめ与えられる CAN のフレームのサイズに一致する．各フレームに対する送信ノードと周期は，いずれかのメッセージがそのフレームに割り当てられた時点で，そのメッセージを送信するノードとその周期に確定される．また，メッセージが割り当てられたフレームの空きサイズは，そのメッセージのデータサイズ分だけ減少する．

解の CAN 順序の値は，すべてのメッセージに対して与えられるが，CAN 上で送信されることのないメッセージについては，この値は意味をなさない．図 7 では，M_0 の CAN 順序の値が 1 となっているが，M_0 は CAN 上で送信されないため，ここでの割当てには関与しない．後に続く M_3，M_4 も同様である．

よって，はじめに F_{c0} に割り当てられるのは M_1 となり，その送信ノードと周期，空きサイズはそれぞれ N_{GW}，8，0 となる．次に，M_2 を考えるが，F_{c0} の空きサイズはすでに 0 であるため，F_{c1} に割り当てられ，その送信ノードと周期はそれぞれ N_{GW}，8 となる．

(5) (h) FlexRay タスクの起動時刻は，解のタスクの優先度と割当て基準，オフセットを用いて決定される．タスクの優先度の値の小さい方から，その起動時刻を決定していく．FlexRay 上の経路の応答時間を小さくするためには，メッセージとその送受信タスクとの間の遅延時間ができるだけ小さくなることが望ましい．そのため，本手法では，タスクの起動時刻をそれらが送信，または受信するメッセージの割り当てられたスロットに基づいて決定する．

割当て基準は，各タスクの起動時刻の決定の際に基準とするメッセージを示している．これらは，各タスクにとって，それが送信，または受信するメッセージのいずれかとなる．割当て基準がタスクの送信するメッセージであった場合，そのメッセージが割り当てられたスロットの開始時刻に，タスクの終了時刻を合わせて配置する．一方，受信するメッセージであった場合には，そのメッセージが割り当てられたスロットの終了時刻に，タスクの起動時刻を合わせて配置する．

このとき，すでに配置されたより優先度の高いタスクによって邪魔される場合，その割当て基準に応じて望ましい方向にずらして配置する．つまり，割当て基準が送信メッセージである場合には，時刻の早い方に，受信メッセージである場合には，時刻の遅い方にずらす．その結果，周期ごとのすべての起動タイミングにおいて，いずれのタスクにも邪魔されない場所が見つかった時点で，そのタスクの起動時刻を決定する．

(3) で述べたダミーメッセージは，割当て基準がノード内メッセージであっても，タスクの起動時刻を決定することができるようにするために割り当てられる．このとき，通常のメッセージとは異なり，スロットの開始時刻にタスクの起動時刻を，スロットの終了時刻にタスクの終了時刻を合わせることによって，連続するタスク間での遅延が生じないように配置できる．

タスク T_i の周期がその割当て基準のメッセージ M_i の周期よりも大きい場合，その配置パターンは P_{T_i}/P_{M_i} だけ存在することになる．解のタスクに与えられたオフセットは，その中で実際に選択するパターンを示している．そしてその値は，各配置パターンに対し，$t = 0$ の時刻から早い順に $0, 1, 2, \cdots, P_{T_i}/P_{M_i} - 1$ のように割り振られている．

図 7 では，はじめに，優先度 1 のタスク T_1 の起動時刻が，その割当て基準メッセージ M_1 のスロットの開始時刻に合わせて，時刻 $t = 1, 9, 17, \cdots$ に決定される．T_3 は CAN 上で処理されるタスクであるため，次に，T_0 の起動時刻を，その割当て基準メッセージ M_0 のスロットの開始時刻に合わせて，時刻 $t = 1, 5, 9, \cdots$ に決定しようとする．しかし，このとき，すでに配置された T_1 に邪魔されるため，時刻の

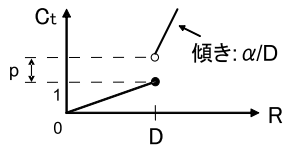


図 8 時間制約コスト
Fig. 8 The cost of the timing constraints.

早い方にずらされ，結局，時刻 $t = 0, 4, 8, \cdots$ に決定される．同様にして，T_2，T_4 の起動時刻も決定される．なお，ここではすべて配置パターンが 1 種類のみであるため，タスクのオフセットはすべて 0 となっている．

(6) (i) CAN タスクの優先度は，解のタスクの優先度に基づき，各ノードについて，その値が小さいタスクから順に高い優先度を割り付ける．図 7 では，CAN 上のノード N_2 で処理されるタスク T_3 と T_5 がそれぞれ優先度 2，5 となっているため，その優先度は高い順に T_3，T_5 となっている．

本手法での近傍解は，元の解に対して以下のいずれかの操作を加えたものとなる．

- 2 つの連続するタスクの優先度の交換
- 1 つのタスクの周期の変更
- 1 つの中間タスクの処理ノードの変更
- 1 つのタスクの割当て基準の変更
- 1 つのタスクのオフセットの変更
- 1 つのメッセージの周期の変更
- 2 つの連続するメッセージの FlexRay 順序の交換
- 1 つのメッセージの FlexRay オフセットの変更
- 2 つの連続するメッセージの CAN 順序の交換

4.3 解の評価

本手法の SA における解のコスト C は，次式で与えられる．

$$C = w_t C_t + w_{lt} L_t + w_{lf} L_f + w_{lc} L_c \qquad (1)$$

ここで，C_t，w_t は時間制約コストとその重み，L_t，w_{lt} はタスクの負荷とその重み，L_f，w_{lf} は FlexRay のネットワーク負荷とその重み，L_c，w_{lc} は CAN のネットワーク負荷とその重みを表している．以降では，これらのコストの計算方法について説明する．

はじめに，時間制約コストの求め方について説明する．与えられたすべての経路に対する時間制約について，個々の解から生成されたスケジュールより，3.2 節で述べた方法を用いて計算した最大応答時間 R と，そのデッドライン D を用いて，**図 8** に示すようなコストを与える．ここでは，デッドライン以下の最大応答時間に対しては，デッドラインでの値を 1 として，最大応答時間に比例させた値を与えている．デッドラインを超える範囲では，その傾きを α（> 1）倍し，さらにペナルティとして一定の値 p を加えている．

解によっては，前節で述べた割当て手順において，割り当てられないタスクとメッセージが存在することがある．そのような場合，それらが割り当てられないことによって求めることのできない時間に対して，それらがとりうる中で最大の値を用いることによって，経路の応答時間を近似的に計算する．これにより，割り当てられないタスクとメッセージの数に応じて応答時間が大きくなり，結果的に時間制約のコストにそのペナルティが反映されることになる．C_t は，時間制約の与えられたすべての経路についてこれを計算し，総和することによって求められる．

次に，タスクの負荷は，全ノードにおけるプロセッサ使用率をタスクの周期と実行時間に基づいて計算し，その和をノード数で割ることによって求められる．FlexRay のネットワーク負荷は，全体のスロット数に対する使用スロット数の割合で与えられる．CAN のネットワーク負荷は，すべてのメッセージの最大送信時間をその周期で割ることによって求められる．これらの負荷にそれぞれ重みを乗じて足し合わせたものを「負荷コスト」と呼び，これを加えることによって，負荷の削減をスケジュールの制約条件として取り入れることを可能にした．

時間制約のコストに対する α と p，および，それぞれのコストに対する重み w_t，w_{lt}，w_{lf}，w_{lc} は，問題の性質やスケジューリングの方針に基づいて決定する．

5. 実　　験

本章では，将来的なアプリケーションを想定したタスクグラフに対して本手法を適用し，評価を行う．本来ならば，本手法と従来手法[7] を同じ対象に対して適用し，その結果に対する比較・評価を行うべきである．しかし 2 章で述べたように，対象とするプロトコルの違いにより両者のスケジューリング方法は大きく異なり，アプリケーションモデルにおける周期の前提条件も異なるため，公正な比較を行うことができない．ここでは，従来の手法において考慮されていなかった負荷の削減や，周期の探索によって得られる効果を実験によって検証し，本手法の有用性を示す．

5.1 負荷の削減効果

本節では，負荷の削減を考慮に入れた効果を示すため，**図 9** に示すタスクグラフを用い，ランダムに生成した 100 パターンの初期解から，負荷コストを加えた場合と加えない場合において，それぞれ本手法を適用した結果を比較する．このタスクグラフは，文

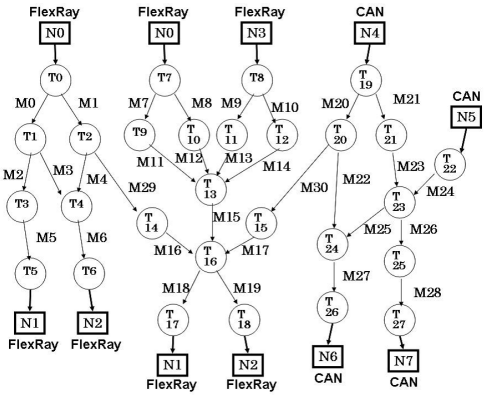


図 9 実験用タスクグラフ
Fig. 9 The taskgraph for experiments.

表 1 負荷の削減効果
Table 1 The effect of reducing the load.

	負荷コストあり	負荷コストなし
FlexRay 負荷 (%)	66.75	80.00
CAN 負荷 (%)	34.22	34.63
GW 通過メッセージ数	1.44	1.89

献 11) において実例として紹介されている 3 つの自動車制御用アプリケーション（adaptive cruise control, electric power steering, traction control）に対応したタスクグラフを参考に，将来的なアプリケーションの拡張を想定して作成したものである．なお，ここでは負荷の削減に対する効果だけを検証するため，周期は探索の対象に含めないものとした．

グラフ内の周期はすべて 2 ms，タスクの最大実行時間はすべてのノードに対し 300 μs，メッセージのデータサイズはすべて 8 バイトとした．時間制約として，各入力タスクから出力タスクへのすべての経路に対し，それぞれ 4〜32 ms の範囲でデッドラインを与えた．FlexRay の通信サイクル長は 2 ms，最大スロット ID は 100，フレームサイズは 16 バイト（通信速度 2.5 Mbps 相当），CAN の通信速度は 1 Mbps とした．

負荷コストを加えた場合と加えない場合で，求められた最良解の FlexRay 負荷，CAN 負荷，GW 通過メッセージ数の全パターンの平均値を**表 1** に示す．

結果より，負荷コストを解の評価基準に加えることによって，同じ制約を課してスケジューリングを行ったときでも，タスクの処理ノードの割り付け方によって GW 通過メッセージ数が減少し，ネットワークの負荷を抑えることができることが確認できた．

5.2 周期探索の効果

次に，周期を探索できるように拡張することで得られる効果を示すため，前節で用いたタスクグラフにおいて，そのネットワークを FlexRay，CAN のそれぞれ単体で構成した場合について，本手法を適用した結果を比較する．周期の探索範囲は 2〜16 ms とし，経路に与えるデッドラインの範囲は 12〜32 ms に変更した．その他の設定は，5.1 節と同様である．

表 2 周期探索の効果
Table 2 The effect of searching the periods.

	FlexRay のみ	CAN のみ
タスク負荷 (%)	41.03	52.50
ネットワーク負荷 (%)	58.05	87.75

FlexRay，CAN のそれぞれでネットワークを構成した場合に，求められた最良解のタスク負荷，ネットワーク負荷の全パターンの平均値を**表 2** に示す．

結果より，FlexRay 単体でネットワークを構成した方が，CAN 単体で構成した場合と比較して，タスク，ネットワークのいずれの負荷も抑えられていることが分かる．これは，FlexRay がすべてのタスクとメッセージのスケジュールを静的に決定できるのに対し，CAN は動的にスケジュールが決定されるためであると考えられる．つまり，CAN 上では各経路の最大応答時間がその周期に大きく左右され，その値も実際に比べて悲観的になってしまうため，同じ時間制約を満たすためでも，FlexRay に比べてその周期を小さくする必要があるからであると考えられる．

このように，負荷の削減を考慮に入れつつ，各タスクとメッセージの周期を探索対象に含めることによって，それぞれのシステム構成，アプリケーションの制約に見合った周期の値を求めることができる．そして，その値は厳密な応答時間の計算方法に基づいてその制約を満たすことを保証しているため，実際のシステムの設計において重要な指針を与える可能性を持つ．

6. おわりに

本論文では，FlexRay と CAN を用いた自動車制御分散システムを対象として，従来手法での個々の決定項目を独立に扱うことの問題点を指摘し，それらを統合的に扱えるようなスケジューリング手法を新たに提案した．また，従来手法で用いられているアプリケーションモデルを拡張し，周期を探索対象に加え，負荷の削減を考慮に入れることによって，より現実的で柔軟なスケジューリングを実現した．最後に，本手法を実際に適用した例を用いて，本手法で得られた結果が実際のシステムの設計において重要な指針を与えうることを示した．

本手法は，将来のシステムの設計を想定して，自動車メーカとの綿密な議論を繰り返し検討を進めてきた

成果である．しかし，その問題設定に関しては，いまだ実用上の課題が残されている．

たとえば，本手法では，メッセージ受信時のフィルタリングにおけるソフトウェア負荷の増大を避けるため，タスクやメッセージの周期の値を制限している．しかし，実際のシステムでは必ずしもそのような前提に対応できるとは限らず，周期の設定方法も含めて前提の妥当性を議論しなければならない．また，現在はノードごとに機能を分割して設計が行われている実際のアプリケーションを，本手法のタスクグラフを用いていかに表現するかも重要な課題である．

このような問題も含めて，今後実際のシステムに対して本手法を適用し，その有効性を検証することが必要となる．また，SA の探索過程やコストに用いたパラメータを，問題やスケジューリング方針に応じて適切に決定する方法や，ネットワーク構成などを含めた新たな探索範囲の拡張などが今後の課題としてあげられる．

謝辞 本研究を進めるにあたり，貴重なご意見をいただいたトヨタ自動車（株）統合システム開発部第一開発室の方々をはじめ，ご協力くださった関係各位に厚く御礼申し上げます．

参 考 文 献

1) Nolte, T., Hansson, H. and Bello, L.L.: Implementing Next Generation Automotive Communications, *Proc. ERTSI'04 in conjunction with RTSS'04* (2004).
2) ISO 11898. Road Vehicles — Interchange of digital information — Controller area network (CAN) for high speed communication (1993).
3) Tindell, K.W., Burns, A. and Wellings, A.J.: Calculating Controller Area Network (CAN) Message Response Times, *Control Engineering Practice*, Vol.3, No.8, pp.1163–1169 (1995).
4) Nolte, T., Hansson, H. and Norström, C.: Probabilistic Worst-Case Response-Time Analysis for the Controller Area Network, *Proc. 9th IEEE Real-Time and Embedded Technology and Applications Symposium* (*RTAS'03*), pp.200–207 (2003).
5) Pop, P., Eles, P. and Peng, Z.: Schedulability-Driven Communication Synthesis for Time Triggered Embedded Systems, *Proc. 6th International Conference on Real-Time Computing Systems and Applications* (*RTCSA'99*), pp.287–294 (1999).
6) Eles, P., Doboli, A., Pop, P. and Peng, Z.: Scheduling with Bus Access Optimization for Distributed Embedded Systems, *IEEE Trans. VLSI Systems*, Vol.8, No.5, pp.472–491 (2000).
7) Pop, P., Eles, P. and Peng, Z.: Schedulability-Driven Frame Packing for Multi-Cluster Distributed Embedded Systems, *ACM Trans. Embedded Computing Systems*, Vol.4, No.1, pp.112–140 (2005).
8) FlexRay Consortium. http://www.flexray.com/
9) Liu, C.L. and Layland, J.W.: Scheduling algorithms for multiprogramming in a hard real-time environment, *J. ACM*, Vol.20, No.1, pp.46–61 (1973).
10) Kirkpatrick, S., Gelett Jr., C.D. and Vecchi, M.P.: Optimization by Simulated Annealing, *SCIENCE*, 220(4598), pp.671–680 (1983).
11) Kandasamy, N., Hayes, J.P. and Murray, B.T.: Dependable Communication Synthesis for Distributed Embedded Systems, *Proc. 22nd Int'l Conf. on Computer Safety, Reliability, and Security* (*SAFECOMP*), pp.275–288 (2003).

(平成 18 年 10 月 10 日受付)
(平成 19 年 2 月 17 日採録)

村上 尚彦

2005 年 3 月名古屋大学工学部電気電子情報工学科卒業．現在，同大学大学院情報科学研究科情報システム学専攻博士前期課程に在学．リアルタイムスケジューリング理論とその応用に関する研究に従事．

冨山 宏之（正会員）

1999 年 3 月九州大学大学院システム情報科学研究科博士後期課程修了．同年米国カリフォルニア大学アーバイン校客員研究員．2001 年（財）九州システム情報技術研究所研究員．2003 年名古屋大学大学院情報科学研究科講師．現在同助教授．SOC や組み込みシステムの設計技術に関する研究に従事．博士（工学）．電子情報通信学会，ACM，IEEE 各会員．

高田　広章（正会員）

名古屋大学大学院情報科学研究科情報システム学専攻教授．1988 年東京大学大学院理学系研究科情報科学専攻修士課程修了．同専攻助手，豊橋技術科学大学情報工学系助教授等を経て，2003 年より現職．2006 年より大学院情報科学研究科附属組込みシステム研究センター長を兼務．リアルタイム OS，リアルタイムスケジューリング理論，組み込みシステム開発技術等の研究に従事．オープンソースの ITRON 仕様 OS 等を開発する TOPPERS プロジェクトを主宰．博士（理学）．IEEE，ACM，電子情報通信学会，日本ソフトウェア科学会各会員．