An Online RBF Network Approach for Adaptive Message Scheduling on Controller Area Networks

MU-SONG CHEN AND HAO-WEI YEN
Department of Electrical Engineering
Da-Yeh University
ChangHua, 51591 Taiwan

The Controller Area Network (CAN) is a communication bus for message transaction in real-time environments. A real-time system typically consists of several classes of messages and a scheduler is responsible to allocate network resources to fulfill timing constraints. Given sufficient bandwidth, the static scheduling algorithms can meet the bounded time delay. However, due to the availability of network bandwidth, not all messages can gain enough bandwidth to achieve the best control performance. Therefore, adaptive message scheduling policies that optimize the bandwidth utilization while supporting timeliness guarantees are of special interest. In this paper, we devise an online adaptive Message Scheduling Controller (MSC), which is designed to dynamically respond to the network dynamics. The MSC is realized by the Radial Basis Function (RBF) network with supervised parameter tuning. Two novel learning algorithms provide complementary effects (i) to prevent possible causes of non-uniform bandwidth allocation and (ii) to reduce possibilities of transmission failures. Simulation results show that the proposed MSC in conjunction with parameter adaptation outperforms the existing Fixed-Priority scheduling and Earliest-Deadline First method, in terms of convergence speed, schedulability, and transmission failure rates.

Keywords - Controller Area Network, Message Scheduling Controller, RBF networks, Earliest-Deadline First

1. Introduction

The Controller Area Network [1][2] is a communication bus for message transmission in real-time distributed environments. Due to its decentralization of control, simplicity, and easy maintenance, CAN has been applied intensively in many time-critical industrial process control and automation systems. Usually, messages in CAN are classified into different classes, depending on their importance. For example, a brake-by-wire system in the vehicle failing to react within a given time interval can cause a fatal catastrophe. While the timeliness of real-time communication must be guaranteed, the delay of non real-time messages, such as diagnostics and management operations, may exceed the desired bound without serious consequences. Moreover, the CAN system can also include different forms of uncertainty between message exchanging. Apparently, the underlying scheduling paradigm has a direct impact on the guarantees for timely behaviors. Therefore, efficient scheduling policies that optimize the limited bandwidth utilization while supporting timeliness guarantees are of special interest.

In general, there are essentially two types of scheduling strategies, i.e. static and dynamic. The simplest static scheduling is the Fixed-Priority scheduling (FPS) [3][4]. However, the FPS does not allow scheduling of dynamic systems. In dynamic scheduling, the decisions about which messages to be transmitted are decided at runtime. Earliest-Deadline First (EDF) scheduling [5] belongs to the class of dynamic algorithms, in which scheduling decisions are based on parameters that can be changed during system evolution. Usually, EDF can prove significantly better than static scheduling with respect to schedulability. Similar report and analysis can refer to [6][7]. Although static and dynamic scheduling algorithms have been proposed to solve message scheduling problems, they are often designed without taking into account the system uncertainty and suffer from time-varying characteristics of network. From the control point of view, both of them belong to open-loop paradigms. In this sense, schedules are not optimally adjusted and are prone to degrade rapidly in overload or time-variant situations. This is in contradiction to the requirement for a flexible system that ought to support system reconfiguration. In this context we have extended our previous results [8] by presenting a complete framework for online scheduling scheme, which includes a dynamic message scheduling controller (MSC) in the closed-loop control. The controller, which is implemented by the radial basis function (RBF) network [9], manipulates input and output data through a set of adjustable parameters. Furthermore, the online learning capability ensures the validity and efficiency of the MSC even when the queue loads are abruptly changed. The major difficulty in the design of the RBF network arises from the incomplete knowledge in applying a supervised learning strategy. Thus, we also suggest two novel learning algorithms, type I and type II learning, for parameter identification.

The rest of this paper is organized as follows. In Section 2, we define the waiting time of messages on the CAN. After presenting the preliminary ideas and notations, a closed-loop control-plant model is present. Then, the RBF network is introduced in Section 3. Afterwards, related online parameter adaptation is also addressed in Section 4. Two novel parameter learning algorithms are thereby presented and employed to provide complementary strategies for message scheduling. In Section 5, experiments are conducted to demonstrate the efficiency and effectiveness of the proposed strategies. Finally, we conclude with a summary and discuss our future research directions in Section 6.

2. Problem Formulation

In this study, we deal with message scheduling problems with multiple classes on the CAN. Before that, a common Message Queue Pool (MQP) in Fig. 1 is assumed to accommodate different classes of messages. Each class of messages is stored in distinct queue for service. To facilitate our analysis, we have to know the temporal model on the CAN. Detail descriptions can refer to [10]. The most crucial factor which dominates the time delay of any messages is the waiting time spending in the queue until it is sent out. In general, the waiting time $W_i(t_k)$ of class i messages at t_k depends on the number of messages in queue (or queue load), priority of messages, and the scheduling strategy, etc.

Intuitively, the longer the message stays in the queue, the larger the waiting time is. To minimize message waiting time, our goals focus on designing a control methodology to maintain bounded time delay. We assume that each class of message is associated with a deadline D_i. In this sense, a message is considered as meeting its time constraint when $W_i(t_k)$ is bounded by the relative deadline. On the other hand, if $W_i(t_k)$ is greater than D_i, a message is not guaranteed to be transmitted successfully. Due to this fact, an important challenge in designing the message scheduling controller is to decide the optimal message transmission sequence (MTS). Accordingly, the problem can be formulated in the following constrained problem

$$\text{Find optimal MTS } \{Q_{i_1}(t_1), Q_{i_2}(t_2), Q_{i_3}(t_3), \cdots\} \qquad (1)$$
$$\text{subject to } D_{j_m} \geq W_{j_m}(t_k),\ \forall\ k, j_m$$

To facilitate our analy

$$\delta_i(t_k) = W_i(t_k) - D_i \qquad (2)$$

The delivery delay provides an important assessment to evaluate the performance of scheduling policy and is used intensively in the rest of this paper. With the definition of $\delta_i(t_k)$, Eq. (1) can be reformulated as

$$\begin{aligned}&\text{Find optimal MTS } \{Q_{i_1}(t_1), Q_{i_2}(t_2), Q_{i_3}(t_3), \cdots\}\\ &\text{subject to } \delta_{i_k}(t_k) \leq 0,\ \forall\ k, i_k\end{aligned} \qquad (3)$$

From Eq. (2) or (3), a message is said to be timely transmitted if $\delta_i(t_k) \leq 0$. On the contrary, a message cannot guarantee its timing constraints or is notified of transmission failure.

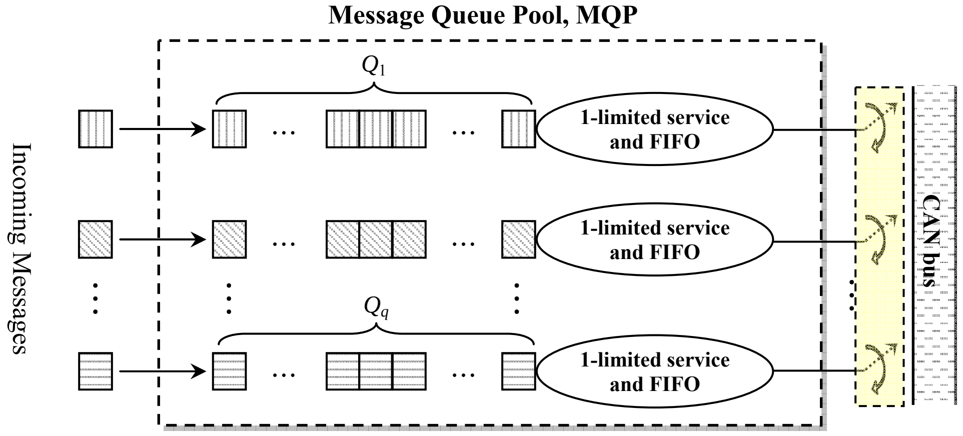
Message Queue Pool, MQP
Incoming Messages
Q_1
Q_q
1-limited service and FIFO
1-limited service and FIFO
1-limited service and FIFO
CAN bus

Fig. 1. The schematic diagram of the MQP where each queue is served in First-In-First-Out (FIFO) discipline.

Unfortunately, the solutions for optimal MTS often result in a combinatorial optimization problem which has been proven NP-hard in [11]. One of the possible approaches is to investigate all feasible solutions which are also known as exhaustive search with exponential runtime complexity [12]. Apparently, as the number of messages increases, the number of possible combinations grows even more rapidly making it impossible to guarantee a feasible solution. Intuitively, the CAN system is often a control system in the sense that the system samples the environment through sensors, calculates some control actions based on these readings, and then propagates the actions back to the environment through actuators. Based on this fact, we propose a discrete-time closed-loop controller-plant model, as the one depicted in Fig. 2. In general, the state space model of Fig. 2 in discrete time domain assumes the form

$$\begin{aligned}\mathbf{x}(t_{k+1}) &= f\left(\mathbf{x}(t_k), \mathbf{y}(t_k)\right) \quad &\text{(plant)}\\ \mathbf{y}(t_k) &= \Phi\left(\mathbf{x}(t_k)\right) \quad &\text{(controller)}\end{aligned} \qquad (4)$$

where $\mathbf{x}(t_k)$ and $\mathbf{y}(t_k)$ are the state vector and control signal, respectively. It is worth noting that vector $\mathbf{x}$ contains measured quantities from the network dynamics and $\mathbf{y}$ is the decision vector from the scheduler. In other words, the message scheduler determines optimal control policies to make a transition of state vector from $\mathbf{x}(t_k)$ to $\mathbf{x}(t_{k+1})$ under control $\mathbf{y}(t_k)$. A major benefit of closed-loop control is to incorporate the time-varying characteristics of network into the design specifications. From the control point of view, the message scheduling controller behaves like a message dispatcher and is responsible for deciding the optimal MTS, whereas the plant is analogous to the MQP. The closed-loop control in Fig. 2 is divided into three stages, i.e. prediction stage and training stage in the MSC and transmission stage in the MQP. In the prediction stage, the MSC is responsible for distributing resources to queued messages as a function of the time-varying nature of the network. In the transmission stage, the messages stored in the winner queue are then retrieved in the First-In-First-Out discipline. The dedicated message is sent out and $\mathbf{x}(t_k)$ forwards to its next state $\mathbf{x}(t_{k+1})$. The process continues until any transmission failures occur. In the training stage, regularity of controller's parameters is critical in order to maintain timeliness transmissions under constraints on the availability of network resources and workload uncertainty.

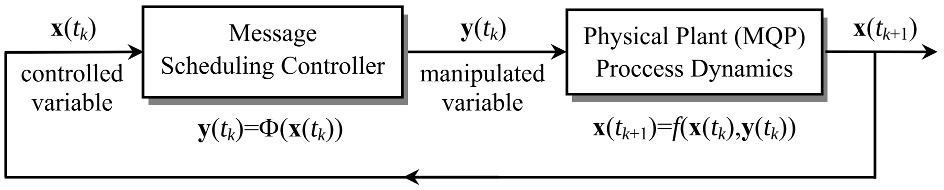
$\mathbf{x}(t_k)$
controlled variable
Message Scheduling Controller
$\mathbf{y}(t_k)=\Phi(\mathbf{x}(t_k))$
$\mathbf{y}(t_k)$
manipulated variable
Physical Plant (MQP) Proccess Dynamics
$\mathbf{x}(t_{k+1})=f(\mathbf{x}(t_k),\mathbf{y}(t_k))$
$\mathbf{x}(t_{k+1})$

Fig. 2. Block diagram for a closed-loop control system on the CAN model.

The key issue of updating controller's parameter requires the explicit knowledge of $f(\cdot)$ and $\Phi(\cdot)$. Usually, $f(\cdot)$ and $\Phi(\cdot)$ denote some nonlinear functions and are generally unknown. In this context, we propose to utilize machine learning (ML) methodology to resolve this problem. With a set of sample data, ML algorithm can regulate controller's parameters to capture the relationships between input and output variables. Therefore, we may not involve a detail mathematical model for the controller. Among many existing ML algorithms, the RBF network is one of the most appealing techniques with flexible structure. The RBF network is capable of identifying complex nonlinear relationships between sample data without attempting to reach understanding as to the nature of the phenomena. Therefore, we adopt the RBF network as our learning platform for message scheduling. In what follows, we focus on the discussion and design of the MSC. We also present two novel learning algorithms for parameter adaptation. Since the RBF network is functionally equivalent to the MSC, both terminologies are used interchangeably in the rest of this paper.

3. Radial Basis Function Network

The RBF network is a single hidden layer neural network [9]. The hidden layer consists of an array of RBF nodes (or hidden neurons) where each node calculates the Euclidean distance between the input vector and parameter vector of the neuron. The resulting value is then passed through a nonlinear function ϕ_j, which is also known as the radial basis function. Throughout this paper, we make use of Gaussian function as the RBF in the following formulation

$$\phi_j(\mathbf{x},\mathbf{c}_j,\sigma_j)=\exp\left(-\frac{1}{2}\frac{\left\|\mathbf{x}-\mathbf{c}_j\right\|^2}{\sigma_j^2}\right) \tag{5}$$

Usually, the outputs o rly by different weights to represent the actual output as

$$y_k(\mathbf{x})=\sum\nolimits_j w_{jk}\phi_j(\mathbf{x},\boldsymbol{\theta}_j) \tag{6}$$

where w_{jk} is the output weight and is connected between ϕ_j and an output unit k. $\boldsymbol{\theta}_j$ are adjustable parameters, such as centers and spreads of ϕ_j. According to the Cover's theorem [13], provided that the transformation from the input space to the feature space is nonlinear and the dimensionality of the feature space is highly compared to that of the input space, there is a high likelihood that a non-separable classification task in the input space is transformed into a linearly separable one in the feature space.

In the development of an optimal RBF network model, the parameter estimation procedure must include (1) the selections of centers $\mathbf{c}_i$ and widths σ_i of ϕ_j and (2) estimations of the weight vector $\mathbf{w}$. From Eq. (6), the free parameter vector $\boldsymbol{\Theta}$ is defined as

$$\boldsymbol{\Theta}=\left\{w_{ij},\mathbf{c}_i,\sigma_i\right\},\quad \forall\ i,j \tag{7}$$

and $\boldsymbol{\Theta}$ should be adjusted depending on the errors. Typically the instantaneous error $E(t)$ is calculated as

$$E(t)=\frac{1}{2}\mathbf{e}^{\mathrm{T}}(t)\mathbf{e}(t) \tag{8}$$

where $\mathbf{e}(t)$=$\mathbf{y}(t)$-$\mathbf{u}_{\text{target}}(t)$ and $\mathbf{u}_{\text{target}}(t)$ is the target values associated with $\mathbf{x}(t)$. In online gradient-based learning method, the goal is to update iteratively the parameter vector $\boldsymbol{\Theta}$ so that $E(t)$ is reduced at each time instant. In such a case, a general parameter update $\Delta\theta(t)$ can be derived from

$$\Delta\theta(t)=-\eta\frac{\partial E(t)}{\partial\theta(t)},\qquad \forall\theta\in\boldsymbol{\Theta} \tag{9}$$

with a small learning rate η.

Regarding the parameter learning of the RBF network, we have to decide (1) how to choose the input quantities, (2) when to initiate the learning process, and (3) how to start supervised parameter adaptation. These issues are discussed in the next section.

4. Parameter Identification

To make the MSC more comparable to our work, we are intended to define appropriate inputs for the RBF network such that the controller becomes more responsive to the network dynamics. Firstly, we consider the instantaneous queue load $\tilde{\rho}_i(t_k)$, which is defined in terms of number of messages $n_i(t_k)$ and the deadline D_i of Q_i

$$\tilde{\rho}_i(t_k)=\frac{n_i(t_k)}{D_i} \tag{10}$$

Accordingly, the instant load $\rho(t_k)$ is the sum of $\tilde{\rho}_i(t_k)$ for all queues

$$\rho(t_k) = \sum\nolimits_{\forall i} \tilde{\rho}_i(t_k) \tag{11}$$

Eq. (11) is analogous to the bandwidth utilization ratio suggested by [4]. It is required that $\rho(t_k)$ is less than or equal to one to guarantee the schedulability of messages. Apparently, a large value of $\rho(t_k)$ can result in a heavy load and a significant growth of waiting time, excluding the CAN to be a real-time network. Certainly, including $\tilde{\rho}_i(t_k)$ in the controller's input becomes more important when any queue load is abruptly changed. This phenomenon is verified in the second experiments in section 5.

Secondly, the delivery delay $\delta_i(t_k)$ in Eq. (2) reveals whether the message is timely transmitted or not. Without this information, the controller contains less knowledge about the message behaviors. It is evident that the delivery delay needs to be taken into consideration in the analysis such that the MSC can react to the network conditions directly. Finally, limited bandwidth occupied by messages is another important quantity for any real-time system. If any message's waiting time nearly approaches its deadline while still satisfying timing constraints, any unexpected changes of queue load can induce unpredicted adversity. To alleviate this problem and provide fairness of bandwidth allocation, we should notice the variations of waiting times for the controller's inputs. We thus calculate the average waiting time $\bar{W}_i(t_k)$ over a sliding window with length M. A sliding window with a length of M measurements can be represented as $\{W_i(t_{k-1-M}),\ldots,W_i(t_{k-1}),W_i(t_k)\}$. Then the average waiting time of class i message is

$$\bar{W}_i(t_k) = \frac{1}{M}\frac{1}{D_i}\sum_{r=k-1-M}^{k} W_i(t_r),\ \forall i \tag{12}$$

and the variation or standard deviation D_σ of $\bar{W}_i$ is

$$D_\sigma(t_k) = \mathrm{STD}(\bar{W}_1(t_k),\bar{W}_2(t_k),\cdots,\bar{W}_q(t_k)) \tag{13}$$

At this point, the input attributes that are of relevance include $\tilde{\rho}_i(t_k)$, $\delta_i(t_k)$, and D_σ. These quantities should contain time-varying characteristics of CAN and are very informative for the controller to "learn" to be an expert for scheduling predictions. We now arrive at our final expression for the inputs of the MSC in the following formulation

$$\mathbf{x}(t_k) = \begin{bmatrix}\chi_1(t_k) & \chi_2(t_k) & \cdots & \chi_q(t_k) & D_\sigma(t_k)\end{bmatrix}^{\mathrm{T}} \in R^{2q+1} \tag{14}$$

where $\chi_i(t_k)$ is the vector with elements $\tilde{\rho}_i(t_k)$ and $\delta_i(t_k)$. In our simulation tests, $\delta_i(t_k)$ is normalized by the deadline, namely $\tilde{\delta}_i(t_k)=\delta_i(t_k)/D_i$. Therefore, $\chi_i(t_k)$ is expressed as $\chi_i(t_k)=[\tilde{\rho}_i(t_k),\tilde{\delta}_i(t_k)]$.

In the training stage, we consider two situations to start parameter adaptation. Firstly, too large of the value D_σ indicates that either some classes of messages seldom gain the opportunity to access the network medium or the bandwidth allocation is not fairly shared among requesting messages. In the long run, the waiting time of some messages increase significantly. This is referred to the phenomena of non-uniform bandwidth utilization. The controller should notify this fact and allocate more bandwidth for those messages to avoid possible transmission failures. In this circumstance, type I learning is invoked for parameter adaptation. Secondly, we should consider the situations whenever any transmission failures really occur. When a transmission failure occurs, the controller needs a prompt recovery through effective parameter adaptation. This is referred to type II learning. In fact, both types of learning provide complementary effects (i) to prevent possible causes of non-uniform bandwidth allocation and (ii) to reduce possibilities of transmission failures. After we have decided two critical situations for invoking training processes, we show how to start type I and type II learning.

A. *Type I Learning*

The standard deviation D_σ in Eq. (13) is used as a measure to show how much the waiting time deviates from its mean value. Usually, a high value of D_σ appears to be due to inappropriate decisions made by the MSC. In this circumstance, we expect to keep D_σ as low and uniform as possible. To this end, when the value of D_σ is greater than a predefined threshold θ_σ, the RBF network should be aware of the tendency of the network situations and assigns more bandwidth for messages in Q_J, where $J = \arg\max_j \bar{W}_j(t_k)$. Accordingly, type I learning is involved. The above concept can be accomplished by taking the target vector $\mathbf{u}_{\text{target}}$ as

$$\mathbf{u}_{\text{target}}(t_k) = \begin{bmatrix} 0 & \cdots & \underbrace{1}_{J\text{th position}} & \cdots & 0 \end{bmatrix}, \quad \text{where } J = \arg\max_j \bar{W}_j(t_k) \tag{15}$$

Thus, the input vector $\mathbf{x}(t_k)$ and the above formulation consist of implicit knowledge for a uniform bandwidth utilization and the internal parameters of the RBF network are updated accordingly. Consequently, the likelihood of the RBF network to produce that particular action, $\{\mathbf{x}(t_k),\mathbf{u}_{\text{target}}(t_k)\}$, is reinforced. The pseudocode in Table 1 describes the operations of type I learning.

Table 1. Procedure of type I learning.

```
Procedure : type I learning
When $D_\sigma > \theta_\sigma$, find $J = \arg\max_j \bar{W}_j(t_k)$
extract target vector (refer to Eq. (15))
loop supervised parameter learning
    forward $\mathbf{x}(t_k)$
    calculate error vector $\mathbf{e}(t_k)=\mathbf{y}(t_k)-\mathbf{u}_{\text{target}}(t_k)$
    Backpropagate error $\mathbf{e}$ and update parameters $\mathbf{\Theta}$ of the RBF network
end
```

B. *Type II Learning*

In response to any transmission failures, i.e. $\delta_i(\cdot) > 0$, type II learning is activated. Although many factors can result in the current transmission failure, it is still very complex to analyze any critical issues in affecting the existing failures. However, the value of $\delta_i(t_k)$ reflects how much a message has been delayed for timely transmission. If $\delta_i(t_k)=r$ for a positive integer r, it implies that there are at least r messages in Q_i cannot satisfy timely transmissions. Instead of remedying these problems, we are intended to minimize the transmission failures in the subsequent time instants. To our best efforts, we should allocate more bandwidth for those delayed messages such that the probability of transmission failures can be minimized. In other words, we expect to keep at most r transmission errors in the next r time instances. To this end, the target vector associated with $\mathbf{x}(t_k)$ is taken to be the following form

$$\mathbf{u}_{\text{target}}(t_k) = \begin{bmatrix} 0 & \cdots & \underbrace{1}_{J\text{th position}} & \cdots & 0 \end{bmatrix}, \quad \text{where } J = \arg\max_j \delta_j(t_k) \tag{16}$$

The value J refers to the message with maximum delivery delay. This is analogous to enhance the failure experience by reinforcing the connection weights linked with the output node J in the RBF network. In this regard, the MSC is taught to learn this cause-and-effect relation. Unfortunately, the crisp values in $\mathbf{u}_{\text{target}}$ always introduce extra side effects, i.e. overfitting, which memorize the training pattern rather than representing the systematic structure of the data. To resolve this problem, $\mathbf{u}_{\text{target}}$ is fuzzified by the class membership. The class membership is taken to be proportional to the waiting time. In this way, vector $\mathbf{u}_{\text{target}}$ is formulated as

$$\mathbf{u}_{\text{target}}(t_k) = \frac{1}{S_W} \begin{bmatrix} \frac{W_1(t_k)}{D_1} & \cdots & \frac{W_J(t_k)}{D_J} & \cdots & \frac{W_q(t_k)}{D_q} \end{bmatrix} \tag{17}$$

where $S_W = \sum_i \frac{W_i(t_k)}{D_i}$.

The exploitation of the fuzzy-valued target in Eq. (17) should resolve the problem of overfitting and expedite the learning process. Although the proposed approach is not too accurate, the output errors can still move downhill and thereby reach the feasible solution region, even though the movement is not necessary in the exact direction of steepest descent. Simulation experiments can validate our viewpoints in terms of transmission failures. The pseudocode in Table 2 describes the operations of type II learning.

Table 2. Procedure of type II learning.

```
Procedure : type II learning
When $\delta_J(t_k) > 0$
extract target vector (refer to Eq. (16) or (17))
loop supervised parameter learning
    forward $\mathbf{x}(t_k)$
    calculate error vector $\mathbf{e}(t_k)=\mathbf{y}(t_k) - \mathbf{u}_{\text{target}}(t_k)$
    Backpropagate error $\mathbf{e}$ and update parameters $\mathbf{\Theta}$ of the RBF network
end
```

Finally, Fig. 3 illustrates a general framework of how the MSC incorporates type I and II learning with parameter adaptation. Three stages, including prediction, transmission, and training stages, are identified with dash boxes. The training stage is the most crucial part for the MSC. In this stage, the MSC not only makes itself online adapt to the changing dynamics, but also provides novel learning strategies for supervised parameter tuning.

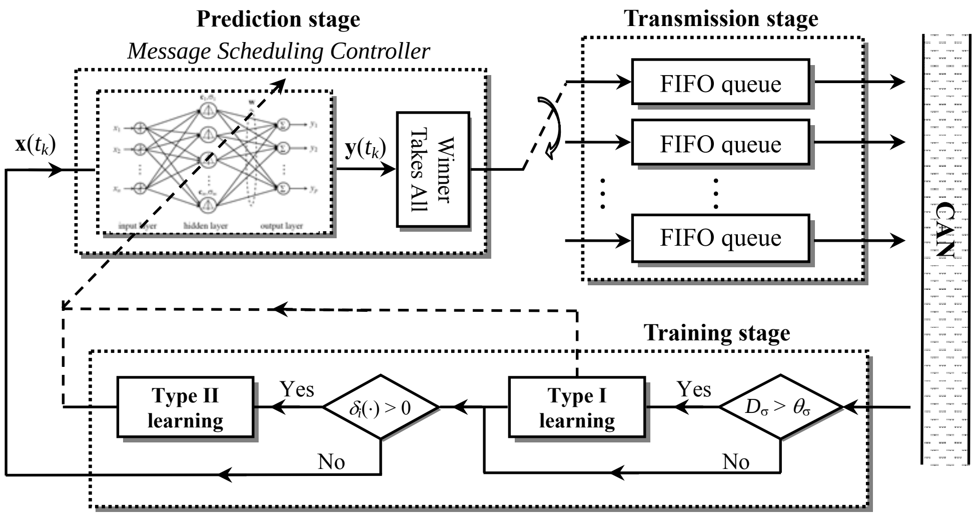

Fig. 3. The complete framework in the design of the MSC for messages scheduling.

5. Simulation

In this section, the applicability of the proposed MSC on CAN message scheduling is demonstrated through a series of simulation experiments. The MSC makes use of type I and II learning for parameter adaptation. Throughout our simulations, there is a set of messages and each message is characterized by $\{D_i, W_i(t_k)\}$, where D_i and $W_i(t_k)$ denote the deadline and waiting time for messages in Q_i, respectively. Our convention is that messages in Q_i have lower D_i (or higher priority) over those messages in Q_j, if $i < j$. Furthermore, we consider the non pre-emptive priority[1] scheduling on the CAN messages. The simulation results are also compared with the FPS and EDF.

Simulation 1

In the first example, we consider three classes of messages (q=3) to be scheduled for a variety of loads, ranging from ρ=0.2 to 0.8. The value of ρ is calculated from Eq. (11). The RBF network is with six RBFs. According to the input formulation in Eq. (14), the layer structure of the RBF network is 7-6-3. Since the tunable parameters are initialized with random seeds, each experiment is repeated 10 times and the final results are taken from their average values. Table 3 summarizes several essential runtime parameters.

Table 3. Runtime parameters[2].

D_i	$\{100\,T, 200\,T, 300\,T\}$
ρ	0.2:0.05:0.8
θ_σ	0.014
structure of the RBF network	7-6-3

[1] With non-preemptive scheduling, any process that is running continues running until finished.
[2] T is the basic time unit to calculate waiting times of messages.

Simulation time	10,000 T
# of repeated experiments	10

Fig. 4 shows the waiting times under conditions of varying loads and different scheduling methods, i.e. MSC, EDF, and FPS. In Fig. 4(a), the MSC curve incurs a longer waiting time for messages of class 1 (i.e. Q_1) and shorter waiting times for the other two classes. It implies that high priority messages in Q_1 can be deferred in favor of low priority messages in Q_2 and Q_3 until they become urgent in (b) and (c). This compromise is considered important in order to maintain a low and uniform distribution of waiting time and thus efficient bandwidth utilization. On the contrary, the waiting time of low priority messages are greatly affected with increasing loads in the case of FPS, especially when ρ is greater than 0.5. This is because that only higher priority messages can gain access to the network at the expense of sacrificing bandwidth of lower priority messages. By contrast, the waiting time of the EDF is somewhat between the MSC and FPS when $\rho > 0.5$. When the load is low ($\rho \leq 0.5$), the EDF has similar waiting time in Q_1 as that of the FPS, due to its deadline expiry scheme. Unlike the FPS method, the EDF dynamically changes messages' priorities when δ_i changes or ρ increases.

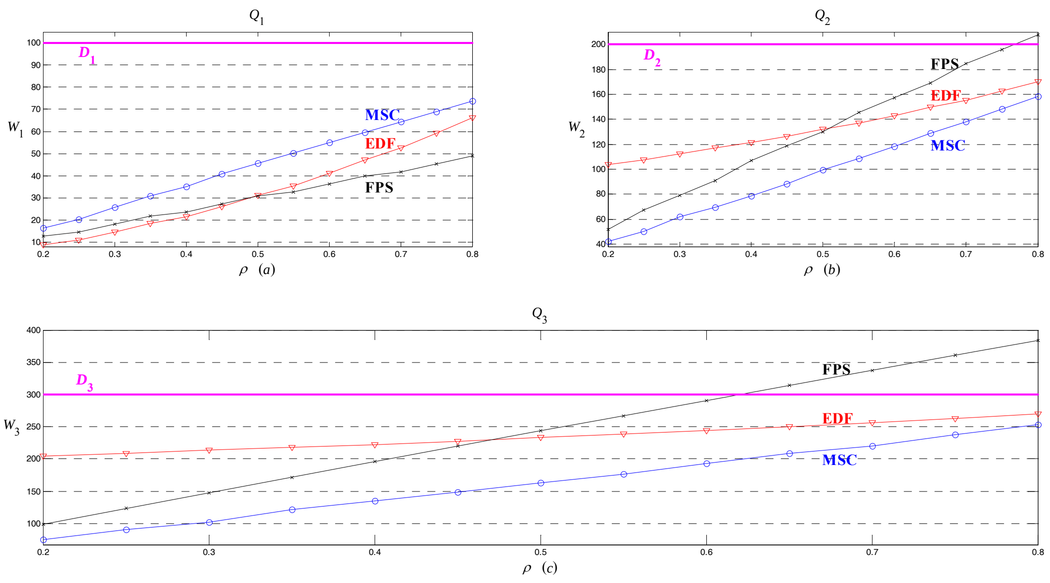

Fig. 4. Average waiting time of messages versus loads and different scheduling methods.

Moreover, we compare the average transmission failure rate (TFR) for different methods. The individual TFR$_k$ of messages in Q_k is defined as

$$\text{TFR}_k = \frac{\text{\# of TFs of message in } Q_k}{\text{\# of message in } Q_k \text{ being transmitted}} * 100 \qquad (18)$$

In Table 4, the TFR of the FPS increases significantly whenever the load is larger than 0.55. On the contrary, both of the MSC and EDF keep almost no transmission failures until ρ is greater than 0.65. After that, both of them reveal similar behaviors except that the MSC still retains a lower TFR than the EDF. These results exhibit the correct functioning of hybrid learning and demonstrate a considerable improvement in the TFRs.

Table 4. Comparisons of average TFRs for the MSC, EDF, and FPS.

ρ	0.2	0.25	0.3	0.35	0.4	0.45	0.5
MSC	0	0	0	0	0	0	0
EDF	0	0	0	0	0	0	0
FPS	0	0	0	0	0	0	0
ρ	0.55	0.6	0.65	0.7	0.75	0.8	
MSC	0	0	0	0.34 %	0.38 %	1.51%	
EDF	0	0	0	0.66 %	2.69 %	3.32 %	
FPS	0	14.06 %	31.33 %	31.44 %	45.5 %	63.2 %	

To verify the effects of type I and type II learning and the fuzzified representation of $\mathbf{u}_{\text{target}}$, we replicate the simulation and illustrate more graphs, e.g. type I learning, type II learning, and hybrid learning with binary outputs (Eq. (16)) and fuzzy outputs (Eq. (17)) in Fig. 5. There is no surprise that the hybrid learning (type I + type II) can achieve lower TFR than that of either type I or type II learning. When only type I learning is involved, it can also result in a lower TFR than type II learning. As was expected, type I learning can only assure timely transmission when ρ is less than 0.55 in this simulation. In hybrid learning, the one with fuzzified outputs can also attain the lowest TFR than the one with binary outputs. This verifies the fact that fuzzified outputs can avoid the sensitivity issues of the backpropagation algorithm.

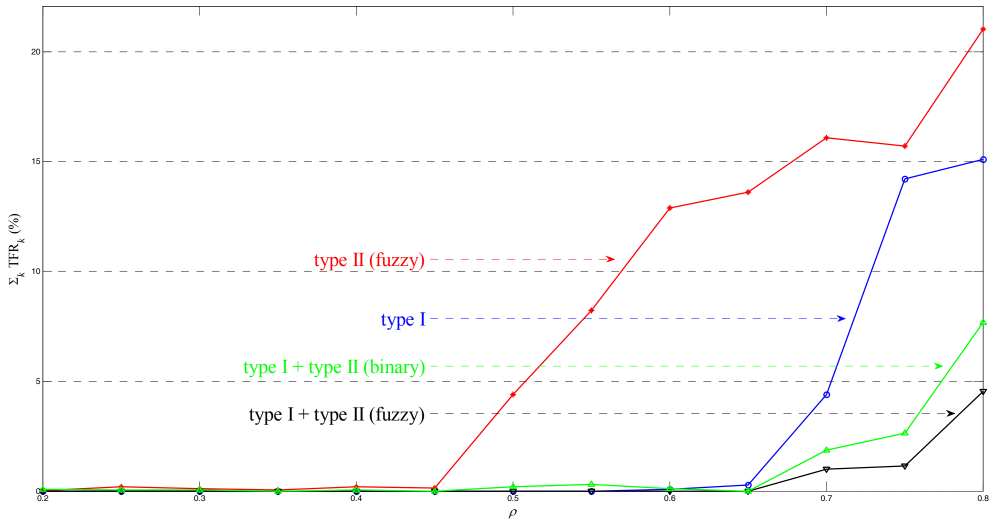

Fig. 5. Evaluation the performances of type I learning, type II learning, and hybrid learning (type I + type II) with binary outputs and fuzzy outputs.

Fig. 6 also demonstrates the standard deviation (STD) curves for waiting times. Among these curves, we observe that the MSC keeps a steady value under different loads. This mainly benefits from the inclusion of D_σ in the input quantities. The steady tendency and low values of STD imply that the bandwidth allocation is in a fair manner.

The advantage of low and uniform STD curves can prevent the possibilities of transmission failures whenever the load becomes saturated or the load is abnormally changed. This importance will show in the next simulation. Interestingly, the behaviors of the EDF and FPS are quite different. In the EDF, when the load is low, the high priority messages can easily gain access to the bus than the low priority messages. Consequently, variations of the messages' waiting times are large and its STD value is high. When the load is high, all messages compete for the shared resources according to the deadline expiry policy. Therefore, the waiting time for messages is comparable and the STD value is low. This explains why the STD curve of the EDF decreases as the load increases. However, the priority of the FPS depends inversely on the deadline and is independent of the load. As the load becomes heavy, the lower priority messages starve from bus access. Hence, its STD curve increases linearly as the load increases. From the figure, one can also note that the STD curves in EDF and FPS are very sensitive to the load change. Finally, the queue selection probabilities P_i for the MSC are recorded. The MSC can retain steady probabilities, i.e. 0.374±0.020, 0.330±0.009, 0.296±0.014, without too much affected by the variations of ρ. Although P_i is nearly constant, the orders of queue selection can be significantly different. In this sense, the limited bandwidth is used optimally by altering the orders of queue service but still keeping a nearly constant probability. This fact clearly explains why the TFRs of the MSC can be lower than the EDF or FPS, as shown in Table 4.

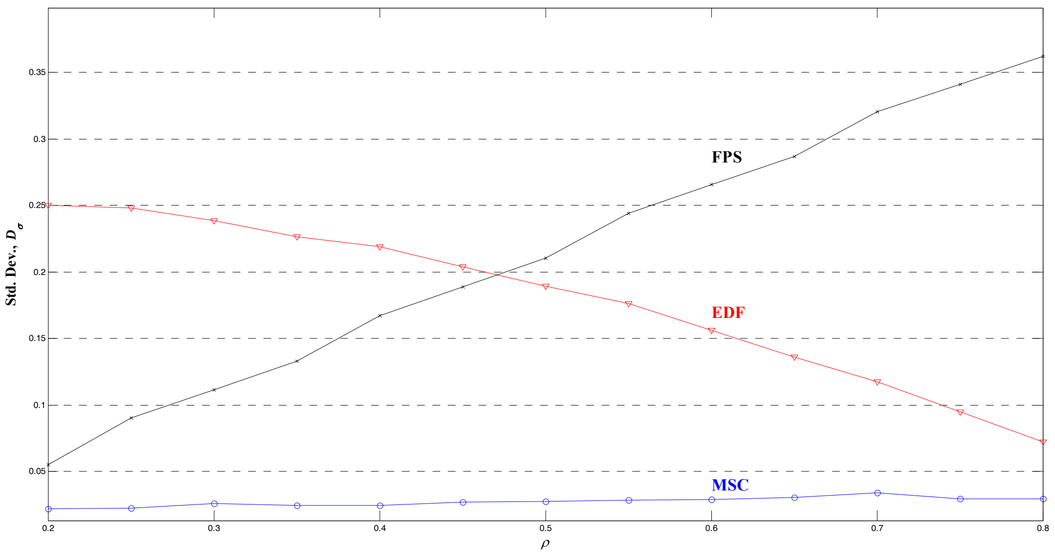

Fig. 6. STD curves for waiting times with different scheduling methods and loads.

Simulation 2

The second example illustrates the flexibility and robustness of the MSC under conditions of transient load. A transient overload occurs when a surge causes network demand to exceed network capacity. In this simulation, all runtime parameters are the same as shown in Table 3 except that the load exhibits uncertain characteristics. Totally, there are three time intervals, i.e. IT_1=[1 *T*, 4750 *T*], IT_2=[4751 *T*, 5250 *T*], and IT_3=[5251 *T*, 10000 *T*]. The load is varied from 0.2 to 0.9 and returns to 0.2 in IT_2. For the sake of clarity, the FPS method is excluded in our analysis, owing to its incompatible performances. In Fig. 7, two vertical dotted lines identify the interval IT_2 and the thick horizontal lines represent the deadlines for messages.

In IT_1 and IT_3, all messages are schedulable and maintain stable waiting times as shown in Fig. 7(*a*), (*b*), and (*c*) under the MSC scheduling. Nevertheless, the waiting times of the EDF in (*e*) and (*f*) exhibit higher fluctuations and instability. These phenomena also explain why the STD curve of the EDF in Fig. 8 is higher than that of the MSC. When ρ is varied from 0.2 to 0.9 in IT_2, this sudden change results in several transmission failures. In an average of ten realizations, the MSC experiences {29.5,42.2,43.6} transmission failures for each class of messages in contrast to {32,76,61} transmission failures for the EDF. Moreover, the STD curve of the MSC returns to its steady value more quickly than that of the EDF. After IT_2, the MSC takes approximately 390 *T* to restore to its steady state as in the IT_1. However, the EDF may require more than 610 *T* to arrive at its stable value. Usually, the EDF does not behave well in the presence of overload, but the MSC can handle overload in a predictable way.

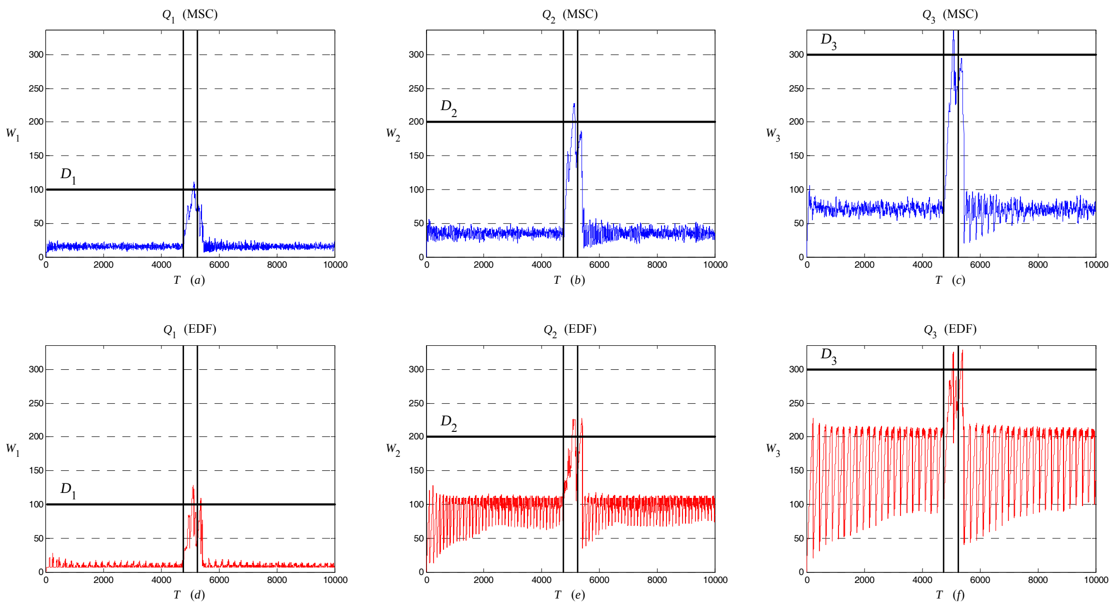

Fig. 7. Average waiting time for the MSC ((*a*), (*b*), (*c*)) and EDF ((*d*), (*e*), (*f*)).

Table 5 also compares the TFRs for both methods in ten runs. The MSC has shown its quick convergence and reliable stability than the EDF scheduling with respect to transmission failures. In summary, it can be observed that the integration of type I and II learning provides complementary effects to sustain a uniform bandwidth allocation and to reduce transmission failures.

Table 5. Comparisons of average TFRs for the MSC and EDF.

run	1	2	3	4	5	6	7	8	9	10
MSC	1.60 %	1.13 %	1.36 %	1.06 %	1.23 %	1.05 %	1.06 %	1.26 %	1.19%	1.35 %
EDF	3.86 %	3.86 %	3.86 %	3.86 %	3.86 %	3.86 %	3.86 %	3.86 %	3.86 %	3.86 %

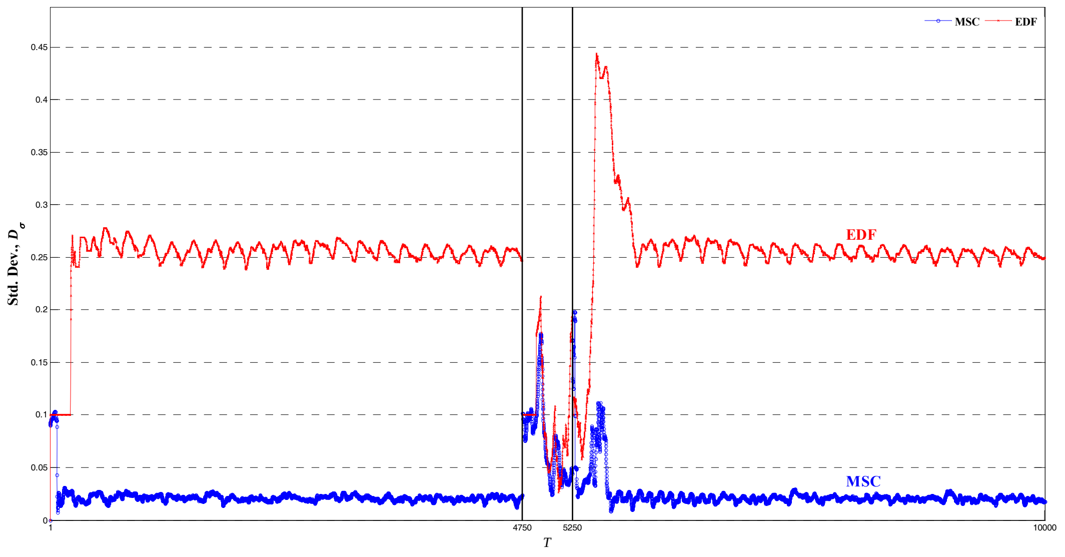

Fig. 8. STD curves of the MSC and EDF.

Finally, we present the computation times, including forward and backward phases of the RBF network. Since the forward phase is necessary for every incoming pattern, it is very straightforward to calculate its CPU time[3]. The average CPU time τ_F of each forward cycle is defined as the ratio of the time spent in forward phase to the actual time involved in the forward learning cycle. Since the forward phase depends mainly on the structure of the RBF network, its computation time is expected to increase with the number of RBFs (n_RBFs). This tendency can be observed from Table 6. In this table, we show τ_F over 10 randomizations. As was expected, τ_F increases approximately from 71 μs to 88 μs when the n_RBFs increases from 1 to 14.

Table 6. Average CPU time τ_F (measured in μs) per forward cycle versus n_RBFs.

n_RBFs	1	2	3	4	5	6	7
τ_F	71.1±11.1	82.6±14.3	83.4±14.6	83.7±14.4	84.1±15.2	84.7±14.5	84.7±14.4
n_RBFs	8	9	10	11	12	13	14
τ_F	85.2±14.6	86.1±14.8	86.4±14.9	86.9±14.8	87.0±14.9	87.6±15.0	88.2±14.9

On the other hand, the CPU time in the backward phase is more complex to estimate.

[3] The CPU time is measured in microseconds using a personal computer with Intel Core2 CPU (6600@2.40GHz with 2G MB memory) and Matlab 7.0.

The CPU time of backward learning cycle depends not only on the n_RBFs but also on the traffic load ρ and θ_σ. Due to limited space, we only consider the case where n_RBFs is varied from 1 to 14 when $\{\rho,\theta_\sigma\}$ is $\{0.8,0.014\}$. Given a RBF network with m RBF nodes and p-class of messages, the total number of tunable parameter is $(2p+2)*m + m*p$, where the first term is the free parameters of RBFs and the second term is related to the number of connection weights. To facilitate our analysis, we define $\tau_{B,\mathrm{I}}$ and $\tau_{B,\mathrm{II}}$ as the average CPU time per learning cycle for both types of learning. $\tau_{B,\mathrm{I}}$ and $\tau_{B,\mathrm{II}}$ can also be decomposed as the product of the average number of iterations (ν_I and ν_II) in each learning cycle and the average CPU time per iteration (λ), i.e. $\tau_{B,\mathrm{I}}=\lambda*\nu_\mathrm{I}$ and $\tau_{B,\mathrm{II}}=\lambda*\nu_\mathrm{II}$. Due to the increasing number of tunable parameters, the average CPU time per iteration is expected to increase. An increasing tendency of λ with n_RBFs can be observed in Fig. 9(*e*). The simulation result is in good agreement with our expectation. Another two graphs in (*c*) and (*d*) are the average number of iterations for type I and II learning. It should be noted that the required parameter iterations are quite small and give rise to fast convergence. In addition to this, the average CPU time per learning cycle is also illustrated in (*a*) and (*b*) for comparisons. In these sub-figures, $\tau_{B,\mathrm{I}}$ is increasing steadily while $\tau_{B,\mathrm{II}}$ is increasing with a little bit fluctuation. The oscillation of $\tau_{B,\mathrm{II}}$ mainly comes from the rising and falling of ν_II in (*d*).

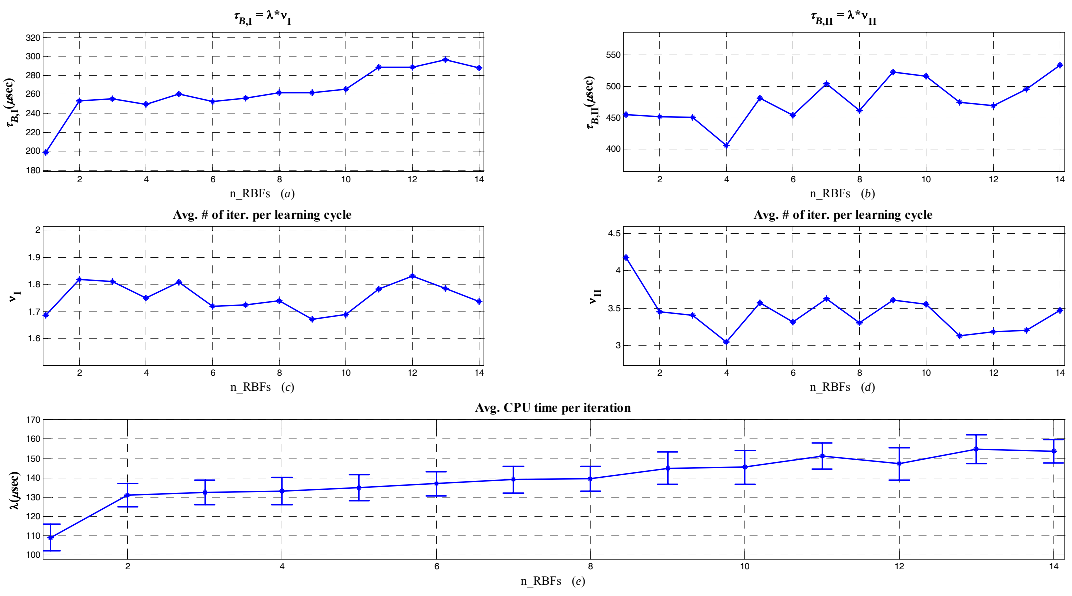

Fig. 9. Average computation time of type I and type II learning in backward learning cycle with the variations of n_RBFs when $\{\rho,\theta_\sigma\}$ is $\{0.8,0.014\}$. (*a*) and (*b*) are the average CPU time for both types of learning. (*c*) and (*d*) are the average iteration number for both types of learning. (*e*) shows the average CPU time required for each iteration.

6. Conclusion

In this paper, an online adaptive message scheduling controller is developed, which can not only perform efficiently in message scheduling but also adapt to the time-varying natures of the CAN system. The controller makes use of feedback information from execution-time measurements and workload changes to regulate its internal parameters. Furthermore, the variations of message waiting time can be kept as uniform and low as possible during system evolution. Simulation results demonstrate that the proposed MSC outperforms the existing scheduling methods. It is also noteworthy that the cooperation of type I and II learning offers very good resistances to the workload variations.

There is, nevertheless, another important issue about the selection of an optimal number of radial basis functions. Too small the number of RBFs leads to a high bias and low variance estimator, whereas too many RBFs yield a low bias but high variance estimator. Accordingly, a structure optimization algorithm should provide an optimal compromise between variance and bias of the estimation made by the parameter training. Our future work should focus on devising an adaptive strategy to modify both the structure of the RBF network and its internal parameters in order to achieve the best performance for message scheduling. Furthermore, to fulfill real-time applications, we are going to integrate the embedded software development flow (i.e. the MSC algorithm) into each CAN node, by use of ARM (Advanced RISC Machine) core-based microprocessors. We expect that the CAN nodes together with powerful microprocessor can give us fast computation capability for processing message scheduling in order to satisfy online real time transmissions.

Acknowledgment

This research was supported by the National Science Council under contract numbers NSC94-2213-E-212-036 and NSC95-2221-E-212-062.

REFERENCES

1. CAN Specification, Version 2.0, Robert Bosch Gmbh. Stuttgart, Germany, 1991.
2. Farsi, M., Ratcliff, K., and Barbosa, M, "An overview of controller area network," Computing and Control Engineering Journal, Vol. 10(3), June 1999, pp. 113-120.
3. J.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, Vol. 2(4), Dec. 1982, pp. 237–250.
4. C.L. Liu and J.W. Layland, "Scheduling algorithms for multi-programming in a hard-real-time environment," *Journal of the ACM*, Vol. 20(1), Jan. 1973, pp. 46–61.
5. K.G. Shin, "Real-Time Communications in a Computer-Controlled Workcell," *IEEE Tranaction Robotics and Automation*, Vol. 7(1), Feb. 1991, pp. 105-113.
6. Paulo Pedreiras and Luis Almeida, "EDF message scheduling on controller area network," *Computing & Control Engineering Journal*, Vol. 13(4), pp. 163-170, Aug 200
7. J.A. Stankovic, M. Spuri, K. Ramamritham, and G.C. Buttazzo, Deadline Scheduling for Real-Time Systems - EDF and Related Algorithms, Kluwer Academic Publishers, 1998.
8. Chuan Ku Lin, Hao-Wei Yen, Mu-Song Chen, Chi-Pan Hwang, and Nguyen Minh Thanh, "A Neural Network Approach for Controller Area Network Message

Scheduling Control," *International MultiConference of Engineers and Computer Scientists*, Hong Kong, June 20-22, 2006, pp. 36-41.

9. S. Haykin. Neural Networks: A Comprehensive Foundation. Macmillan College Publishing Company, New York, 1994.
10. Feng-Li Lian, James R. Moyne, and D. M. Tilbury, "Ethernet, ControlNet, DeviceNet: Performance Evaluation of Control Networks," *IEEE Control Systems Magazine*, Vol. 21(1), Feb. 2001, pp. 66-83.
11. C. Oliveira, P. Pardalos, and T.M. Querido, "Integer formulations for the message scheduling problem on controller area networks," in Grundel, D., Murphey, R. and Pardalos, P., Editors, *Theory and Algorithms for Cooperative Systems*, Kluwer Academic Publishers, Dordrecht, 2004, pp. 353-365.
12. S. Mertens, "Exhaustive search for low-autocorrelation binary sequences," Journal of Physics A: Mathematical and General, 29, L473-L481, 1996.
13. T.M. Cover, "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition," *IEEE Transaction Electronic Computers*, Vol. 14, 1965, pp. 326-334.

Mu-Song Chen (陳木松) received his Ph.D. degree in Electrical Engineering from University of Texas, at Arlington in 1992. Currently, he is an associate Professor of the Department of Electrical Engineering of DaYeh University, Changhua, Taiwan, ROC. His areas of research include neuro-fuzzy network, pattern recognition, image processing and controller area network

Chun-Chao Yeh (顏豪緯) is a PhD candidate at the Department of Electrical Engineering, DaYeh University, Changhua, Taiwan, ROC. His current research focuses on designing the message scheduling on controller area network by using machine-learning algorithms and adaptive queueing strategies.