In Proc. of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Seoul, South Korea, August 2012.
Revised version.

CSS: Conditional State-based Scheduling for Networked Control Systems

Xi Chen*, Akramul Azim†, Xue Liu*, Sebastian Fischmeister†
* School of Computer Science, McGill University,
Email: xi.chen7@mail.mcgill,ca, xueliu@cs.mcgill.ca
† Department of Electrical and Computer Engineering, University of Waterloo
Email: {aazim,sfischme}@uwaterloo.ca

***Abstract*—Modern industrial networked control systems (NCSs) tend to be complicated and have dynamic workload by holding a variety of applications via a shared network. The static network scheduling algorithms fit most NCSs due to their deterministic characteristics and timing guarantees, but they cannot handle dynamic workloads for lack of making on-the-fly decisions. The conditional state-based scheduling adds the dynamism in the static scheduling algorithms by automata or more explicitly state chart like formalisms with conditional transitions. In this paper, we propose CSS scheme that applies the conditional state-based scheduling to dynamically schedule different applications in the industrial NCSs. CSS aims at the time-triggered network in the NCSs and uses time division multiple access (TDMA) method to let the applications access the network. To enhance the scalability of the NCSs, we design CSS as a decentralized scheme where each application in NCSs has a local scheduler to make its schedule decisions. Appropriate algorithms are applied to ensure the scheduling decisions made by the local schedulers are consistent and the desired system performance can be achieved. Simulation results demonstrate the effectiveness of the proposed scheme compared to the static TDMA used in real-time networks.**

***Index Terms*—NCSs, conditional state-based scheduling, dynamic TDMA, decentralized scheduling.**

I. Introduction

Recent years have seen the increasing demand of the networked control systems (NCSs) in industry such as automotive, factory automation, avionics, and robotics that require high reliability and efficiency. By introducing the network as a shared communication medium among the transmission components like sensors, controllers and actuators in the control applications, the system performance of the NCSs can be influenced by network-induced transmission delay. How to design the network scheduling to properly allocate the network bandwidth among the control applications in NCSs is important. Traditional network scheduling like the rate monotonic scheduling [1] or round robin are static, and they work in the NCSs with predefined workload and can provide safety as well as timing guarantees. However, modern industrial NCSs tend to be complicated with abundant applications co-existing in the network, which may result in the dynamic workload. One typical case is the automobile market that launches a variety of new car models with increasing functionalities to cater the user demands. Besides the critical control applications in the car like the active steering and drive control that should be executed periodically, the other non-critical applications such as window open/close or DVD player in the network may activate aperiodically, causing workload variations. Under the dynamic workload, scheduling all the applications statically may use the network resource inefficiently and degrade the system performance.

To handle the dynamic workload in NCSs, scheduling algorithms making on-the-fly decisions are the topics-of-interest. Several dynamic scheduling algorithms [2]–[5] were proposed towards the priority-based field bus like Control Area Network (CAN). However, there are not many research efforts on the dynamic network scheduling design for NCSs that use time-trigged network as the shared communication medium. The time-triggered networks like TTCAN [6], FlexRay [7], [8] and real-time Ethernet [9] emerge in industry recently and has the potential to replace the priority-based field bus in future due to its deterministic characteristics and timing guarantees. Some of the time-triggered networks support dynamic scheduling partially. For example, FlexRay and TTCAN reserve a dynamic segment with limited time slots in every communication cycle to schedule the aperiodic applications. Some of the time-triggered networks support dynamic scheduling completely. For example, the real-time Ethernet can make flexible schedule decisions in each time slot. Comparing both types of time-triggered networks, the latter provides better flexibility and scalability. We focus on the NCSs with a time-triggered network that can support dynamic scheduling completely to design the network scheduling algorithm.

The conditional state-based scheduling scheme [9]–[11] represents recent development to improve the network scheduling in real-time systems. The conditional state-based schedules are realized by automata [12] or more explicitly state chart like formalisms with conditional transitions [10], [11]. Network code language permits developers to express such conditional state-based communication schedules, and the specification, analysis, and verification are examined. Therefore, the conditional state-based scheduling inherits the deterministic property of the static scheduling but has more flexibility than the static scheduling to handle the varying workload by adding dynamisms.

In this paper, we propose CSS, which is a conditional state-based scheduling scheme exclusively for industrial NCSs. Given an NCS with multiple applications and a time-triggered network as the shared communication medium, the goal of CSS is to achieve a good overall system performance of the NCS while providing the worst-case guarantees, even under the existence of the dynamic workload or message transmission failure. To enhance the scalability of the NCSs, CSS leverages a distributed consensus to schedule all the

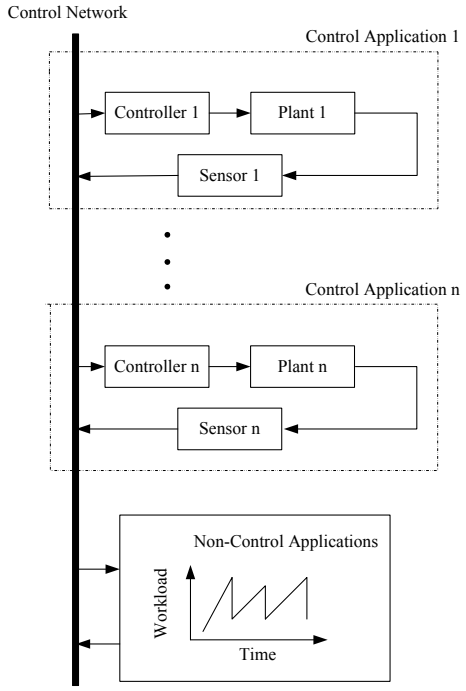


Fig. 1: Networked Control System

applications. Each application in NCSs has the local scheduler to make its schedule decision. The schedule decisions are made according to the specific condition, which is designed to ensure the stability and improve the performance of the critical applications. Moreover, the schedule decisions let the non-critical applications access the network in a best-effort way. We design the message communication principle in NCSs so that the schedule decisions made by the local schedulers are consistent. We simulate a three-servo target-tracking system as a case study, and the results demonstrate the effectiveness of the proposed schedule compared with the static scheduling.

The organization of the paper is as follows: Section II provides the system model of NCSs with time-triggered network. Section III gives details of CSS scheme design. Section IV theoretically analyze the stability of NCSs under CSS scheme. Section V evaluates the performance of the proposed scheme. Section VI introduces the related work. Section VII summarizes the paper.

II. Overview of NCSs with Time-Triggered Network

In our NCS as shown in Fig. 1, all the applications are connected via a time-triggered network. The applications in NCSs consist control and non-control applications. We assume all the control applications like the steering and break control in the car are periodic and critical, and all the non-control applications like the maintenance, diagnostic and entertainment applications are aperiodic and non-critical. Each control application in our NCS has a linear plant, a controller designed in the continuous time domain, and we assume the actuator is directly integrated in the plant. The sensor of each plant samples the plant's states and sends them in one message to the controller via the shared network. The controller then computes the control input corresponding to the sensor's sample and sends it directly to the plant for execution without using the shared network. Since the control application dynamics are considered continuous, the access interval of the control application to the network is much larger than the sampling period of the plant's sensor and the processing time of the controller. Therefore, we assume the sensor can transmit the latest plant's data to the controller as long as it accesses the network, and the controller can react immediately when receiving the messages from the network. To ensure the stability of the control application, the maximum access interval of the control application to the network should be bounded.

The time-triggered network in the NCS is configured with a global clock. All the transmission components in the network have synchronized clocks. We use transmission components to denote all the components in NCSs that sends or receives messages via the network like the sensor and controller in each control application. The network uses TDMA to schedule messages from the transmission components. At each time slot, only the transmission components from one application can access the network and transmit messages. We assume the message transmission time is short and can be bounded by one time slot s.

A. Applications in NCS

Consider an NCS with n control applications and m non-control applications. Each application has a unique index i, $i = 1, \ldots, n+m$. We assign indexes from 1 to n to control applications, and each control application is denoted by C_i, $i = 1, \ldots, n$. We assign the indexes from $n+1$ to $n+m$ to the m non-control applications, and each non-control application is denoted by E_i, $i = n+1, \ldots, n+m$. All the applications compete for using the network.

Each control application C_i, $i = 1, \ldots, n$ can be formulated as a state-space function:

$$\begin{array}{l} \dot{x_i}(t) = A_i x_i(t) + B_i u_i(t) \\ u_i(t) = -K_i x_i(t) \end{array} . \tag{1}$$

The system state is $x_i(t) \in \mathbb{R}^v$. The control input is $u_i(t) \in \mathbb{R}^d$. The state matrix A_i, the input matrix B_i, the feedback matrix K_i have corresponding dimensions and are given at the design time. The closed-loop state matrix of C_i is $\bar{A}_i = A_i - B_i K_i$.

The quality of control (QoC) of each control application C_i depends on two parameters: the control error and the stability. We define the control error $e_i(t)$ as $e_i(t) = x_i(t) - x_i(t_p)$, where t_p is the time point when C_i uses the network at the pth time, and $p = 1, 2, \ldots$. When $t \in [t_p, t_{p+1}]$, $e_i(t)$ is expressed as

$$\dot{e_i}(t) = A_i e_i(t) + \bar{A}_i x_i(t_p). \tag{2}$$

From Eq.(2), we derive

$$e_i(t) = \bar{A}_i x_i(t_p)(t - t_p) + \int_{t_p}^{t} A_i e_i(s) ds. \tag{3}$$

The control error has an exponential relation with the time delay for the control application to access the network. We

define the scaled control error as $\frac{\|e_i(t)\|}{\|x_i(t_p)\|}$, and the time delay for C_i to access the network is

$$l_i = t - t_p, t \in [t_p, t_{p+1}]. \tag{4}$$

From Eq.(2,3) and according to [13], we get

$$\frac{\|e_i(t)\|}{\|x_i(t_p)\|} \leq \frac{\left\|\bar{A}_i\right\|}{\|A_i\|}(e^{\|A_i\|(t-t_p)} - 1). \tag{5}$$

From Eq.(3,4,5) we have the scaled control error $\frac{\|e_i(t)\|}{\|x_i(t_p)\|} = 0$ when $l_i = 0$, and $\frac{\|e_i(t)\|}{\|x_i(t_p)\|}$ increases with l_i exponentially when $l_i > 0$. Since the scaled control error reflects the control performance, we define the performance metric function $J_i(l_i)$ as

$$J_i(l_i) = \begin{cases} 0, l_i = 0, \\ \frac{\left\|\bar{A}_i\right\|}{\|A_i\|} e^{\|A_i\| l_i}, l_i > 0. \end{cases} \tag{6}$$

Besides the performance metric of C_i, the stability is closely related to the time delay l_i. To ensure the stability of C_i, l_i can never exceed its upper bound h_i. We should ensure

$$l_i \leq h_i, \tag{7}$$

where h_i can be computed from [13], [14]. We rewrite Theorem 2.15 in [13] for computing upper bound on the access interval as follows:

Theorem II.1. *Define h_{max} as the maximum access interval set at the NCS design stage for control application C_i and h_{max} satisfies*

$$h_{\max} < \frac{1}{\|A_i\|} \ln\left(\frac{\lambda_{\min}(Q_i)\|A_i\|}{2\lambda_{\max}(P_i)\|B_i K_i\|\left\|\bar{A}_i\right\|} + 1\right), \tag{8}$$

where P_i and Q_i are symmetric positive definite matrices such that

$$\bar{A}_i^T P_i + P_i \bar{A}_i = -Q_i, \tag{9}$$

and $\lambda_{min}(Q_i)$ is the minimum eigenvalue of Q_i while $\lambda_{max}(P_i)$ is the maximum eigenvalue of P_i.

Define the polynomial

$$\begin{aligned} p(x) = {} & \lambda_{\max}(P_i)(R_i^2 + 2R_i)x^4 - 2\lambda_{\max}(P_i)(R_i^2 + 2R_i)x^3 \\ & + [\lambda_{\max}(P_i)R_i^2 - \tfrac{\lambda_{\min}(Q_i)}{2\left\|\bar{A}_i\right\|}]x^2 + \tfrac{\lambda_{\min}(Q_i)}{2\left\|\bar{A}_i\right\|}, \end{aligned} \tag{10}$$

where $R_i = \frac{\|B_i K_i\|(e^{\|A_i\| h_{\max}} - 1)}{\|A_i\|}$. Let x^* be the real root of $p(x)$ greater than 1 (if exists).

The system is exponentially stable if

$$h_i = \begin{cases} \min\left\{h_{\max}, \frac{1}{\left\|\bar{A}_i\right\|} \ln x^*\right\}, \text{ if } x^* \text{ exits}, \\ h_{\max}, \text{ otherwise} \end{cases} \tag{11}$$

where h_i is the upper bound of the delay l_i.

Since C_i's controller is designed ignoring the network and can ensure the stability of C_i in the continuous time domain, there always exist P_i and Q_i that satisfy Eq.(9). Therefore, the existence of h_i can be ensured. Proof of Theorem II.1 refers to [13].

There are non-control applications in the network. Each non-control application E_i, $i = n+1, \ldots, n+m$ aperiodically activates and does not have an explicit deadline.

B. Bandwidth Utilization of The Applications in NCSs

Each application in NCSs uses a portion of network bandwidth during the runtime. We use u_i to denote bandwidth utilization of each application with index i, $i = 1, \ldots, n+m$, which is the percentage of the network bandwidth taken by this application. Each control application C_i, $i = 1, \ldots, n$ should be given at least one time slot in a time span h_i to stay stable, hence the lower bound for the bandwidth utilization of C_i is $\underline{u}_i = \frac{1}{\left\lfloor \frac{h_i}{s} \right\rfloor}$, where $\lfloor v \rfloor$ is to get the maximum integer less than or equal to v. We estimate the worst-case bandwidth utilization of each non-control application E_i at design time, denoted as $\bar{u}_i$, $i = n+1, \ldots, n+m$. If E_i sporadically activates with a minimum inter-activate period d_i, we estimate $\bar{u}_i = \frac{1}{\left\lfloor \frac{d_i}{s} \right\rfloor}$. If E_i activates randomly and the probability E_i activates in each time slot follows a distribution like Poisson distribution or Pareto distribution, we can use the mean value of the distribution plus its variance as $\bar{u}_i$.

In NCSs, we assume the non-control applications are less safety-critical than the control applications. For example, the window open/close is less important than the steering control in a car. At the design time, we should make sure the total bandwidth utilization of the non-control applications in the worst case does not exceed a threshold that jeopardizes the stability of the control applications,

$$\sum_{i=n}^{n+m} \bar{u}_i \leq 1 - \sum_{i=1}^{n} \underline{u}_i. \tag{12}$$

We can use the admission control at the design time to prevent the non-control applications from joining the network if the bandwidth utilization constraint in Eq.(12) is violated. However, even when the bandwidth utilization constraint in Eq.(12) can be satisfied, it is not sufficient to provide stability guarantees of the control applications. To ensure the stability of the control applications, the network scheduling algorithm should be properly designed.

III. Network Scheduling Design

We use the conditional state-based scheduling scheme as the underpinning mechanism to construct CSS scheme which is dedicated for scheduling control and non-control applications in NCSs. Each application has the local scheduler to deicide whether all the transmission components in this application can use the network or not at each time slot. We develop the algorithms to ensure the schedule decisions made by the local schedulers are consistent. The goal of the proposed network scheduling scheme is to provide a good overall system performance while guarantee the stability of the control applications.

A. System Configuration

We design CSS for NCSs that uses time-triggered network. We assume the communication medium provides a reliable atomic broadcast service; therefore, either all transmission components receive a message or none of them do when message fails to be transmitted. Moreover, we assume that the network supports the priority-based arbitration. For the network that does not support the arbitration mechanism, we

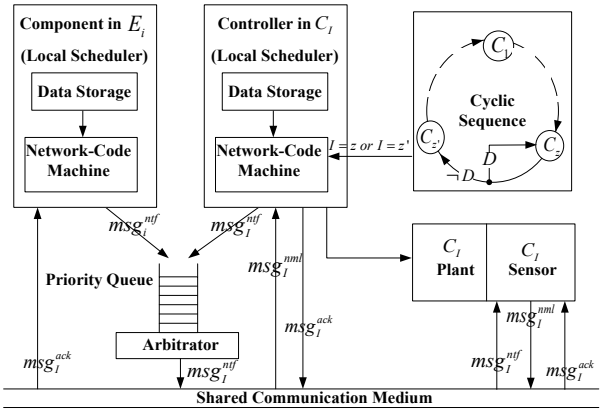


Fig. 2: Framework of CSS Scheme

add an independent processor in the network to act as the network arbitrator.

CSS leverages a distributed consensus to schedule all the applications in NCSs. Each application in NCSs has its local scheduler to implement the conditional state-based scheduling. To avoid additional hardware cost, we usually use an existing transmission component in an application to act as the local scheduler. For example, we choose the controller in each C_i as the local scheduler of C_i. The local scheduler in each application decides whether all the transmission components in the application can use the network or not at each time slot. An existing transmission component is extended to a local scheduler by assigning data storage (i.e. RAM of the processor) for storing scheduling-relevant data and by installing Network-Code Machine (NCM), a programme responsible for making scheduling decision. In each time slot, the local scheduler invokes NCM to update the variables in the data storage and check whether these variables satisfy a specific condition or not. The local scheduler will inform all the other transmission components in the application to use the network only if the specific condition is satisfied.

B. Framework of CSS

CSS allows the developer to define a static schedule for the control applications that ensures the stability of all the control applications without the existence of the non-control applications. Since the non-control applications exist and aperiodically activate in NCSs, CSS allows the developer to define the specific condition that can be leveraged by the local schedulers to make on-the-fly changes to the predefined schedule. The specific condition is designed to improve the overall control performance of the control applications and at the same time to schedule the non-control applications in a best-effort way. The framework of CSS is shown in Fig. 2.

1) Scheduling the Control Applications: We adopt the static TDMA with cyclic sequence as the predefined schedule for all the control applications in NCSs, where all the control applications take turns to use the network and each $C_i, i = 1, \ldots, n$ has its predecessor $C_{'i}$ and successor $C_{i'}$. A communication cycle is the time span during which all the control applications are scheduled according to the cyclic sequence once. The predefined schedule assigns only one time slot for each C_i to access the network in each communication cycle. Therefore, the access interval for each control application C_i to use the network is n time slots, and we set the time slot length s as a value to ensure the stability of all the control applications under the predefined schedule,

$$s < \frac{\min\limits_{i\in\{1,\ldots,n\}} h_i}{n}. \tag{13}$$

The static schedule guarantees fairness to schedule the control applications but is brittle to handle the dynamic workload caused by the non-control applications. As an improvement, CSS scheme can make conditional transitions on the predefined schedule. In each communication cycle, CSS scheme flexibly adds some time slots to schedule the control applications with large control error or the activated non-control applications. Such added time slots are denoted as the extra time slots. For example, if more than one time slots assigned to a control application C_i in a communication cycle, the time slots other than the first one are extra time slots. Moreover, we extend the concept of "extra time slots" to include the slots assigned to the non-control applications for message transmission and the slots with message transmission failure. At each time slot during the runtime, we want to improve the system performance by choosing a control application to use the network. The system performance Q is the overall performance cost of the control applications,

$$Q = \sum_{i=1}^{n} J_i(l_i) = \sum_{i=1}^{n} \frac{\left\|\bar{A}_i\right\|}{\|A_i\|} e^{\|A_i\| l_i}. \tag{14}$$

The smaller Q indicates the better performance. We use $Q_i(k)$ to denote the system performance at the kth time slot ($k = 1, 2, \ldots, \frac{\text{Total Runtime}}{s}$) by choosing C_i to schedule. To improve the system performance under the cyclic structure, CSS needs to schedule the one that can result in the smaller $Q_i(k)$ from two adjacent control applications. Suppose we are currently at the kth time slot, and C_z is the "current control application", which is the control application currently using the network or most recently accessed the network (if the current time slot, namely the kth time slot, is assigned to a non-control application). The local schedulers in C_z and $C_{z'}$ (the successor of C_z, $z' = (z \ mod \ n) + 1$) should decide which control application of them can use the network in the $(k+1)$th time slot based on the following constraint

$$Q_z(k+1) - Q_{z'}(k+1) < 0. \tag{15}$$

If the constraint in Eq.(15) is not satisfied, $C_{z'}$ will be scheduled. Otherwise, we want to schedule C_z. However, scheduling C_z will introduce an extra time slot in the current communication cycle. Under the cyclic structure, the access interval of a control application C_i to the network is at least n time slots. Any extra time slot inserted during the time delay of C_i will prolong the access interval (n time slots) by 1 time slot, which may jeopardize the stability of C_i. We denote the number of the extra time slots in the time delay of C_i as b_i, where $b_i = 0$ when C_i accesses the network with successful message transmission, and $b_i = b_i + 1$ when an extra time slot is introduced during the access delay of C_i. When we prefer

to schedule C_z at the $(k+1)$th time slot, we need to check the following constraint

$$(n + b_i^{(k)} + 1)s \leq h_i, i = 1, \ldots, n \tag{16}$$

to ensure the stability of all the control applications. Eq.(16) is derived by replacing l_i with access interval $(n + b_i^{(k)} + 1)s$ of C_i in Eq.(7). Only when the constraints in Eq.(15) and Eq.(16) are both satisfied can we schedule C_z at the $(k+1)$th time slot.

The schedule decisions are made by the local schedulers according to the performance constraint defined in Eq.(15) and the stability constraint defined in Eq.(16). The performance constraint in Eq.(15) involves the computation to the performance cost of all the control applications, which can be simplified. We use variable I to store the index of the "current control application"and use $l_i^{(k)}$ as the time delay l_i (defined in Eq.(4)) at the kth time slot. Under the cyclic sequence, we use o as the index of the communication cycle where the kth time slot lies in, and we have

$$l_i^{(k)} = \begin{cases} (I - i + b_i^{(k)})s, i \leq I, \\ (I + b_i^{(k)})s, i > I \,\&\, o = 1, \\ (n - i + I + b_i^{(k)})s, i > I \,\&\, o > 1. \end{cases} \tag{17}$$

Consider the situation when k is not in the first cycle ($o > 1$) and I equals to z at the kth time slot. If C_z is chosen to be scheduled at the $(k+1)$th time slot, I will remain to be z and $b_i^{(k+1)}$ will become $b_i^{(k+1)} = \begin{cases} 0, i = z \\ b_i^{(k)} + 1, i \neq z \end{cases}$. If $C_{z'}$ is chosen to be scheduled at the $(k+1)$th time slot, I will become z' and $b_i^{(k+1)}$ will become $b_i^{(k+1)} = \begin{cases} 0, i = z' \\ b_i^{(k)} + 1, i \neq z' \end{cases}$. From Eq.(17), the constraint in Eq.(15) is equivalent to

$$\begin{aligned} & Q_z(k+1) - Q_{z'}(k+1) \\ &= [\frac{\|\bar{A}_{z'}\|}{\|A_{z'}\|} e^{\|A_{z'}\|(n+b_{z'}^{(k)})s} + \sum_{i \notin \{z,z'\}} \frac{\|\bar{A}_i\|}{\|A_i\|} e^{\|A_i\|(l_i^{(k)}+s)}] \\ &\quad -[\frac{\|\bar{A}_z\|}{\|A_z\|} e^{\|A_z\|(b_z^{(k)}+1)s} + \sum_{i \notin \{z,z'\}} \frac{\|\bar{A}_i\|}{\|A_i\|} e^{\|A_i\|(l_i^{(k)}+s)}] \\ &= \frac{\|\bar{A}_{z'}\|}{\|A_{z'}\|} e^{\|A_{z'}\|(n+b_{z'}^{(k)})s} - \frac{\|\bar{A}_z\|}{\|A_z\|} e^{\|A_z\|(b_z^{(k)}+1)s} < 0. \end{aligned} \tag{18}$$

The constraint in Eq.(18) can be further simplified as

$$ln(\frac{a_{z'}}{a_z}) < (\|A_z\|(b_z^{(k)}+1) - \|A_{z'}\|(n + b_{z'}^{(k)}))s, o > 1, \tag{19}$$

where $a_{z'} = \frac{\|\bar{A}_{z'}\|}{\|A_{z'}\|}$ and $a_z = \frac{\|\bar{A}_z\|}{\|A_z\|}$ can be derived at the design time. Consider the situation when k is in the first cycle ($o = 1$), by using a similar deduction as given above, the constraint in Eq.(15) can be simplified as

$$ln(\frac{a_{z'}}{a_z}) < (\|A_z\|(b_z^{(k)} + 1) - \|A_{z'}\|(z + 1 + b_{z'}^{(k)}))s, o = 1, \tag{20}$$

where $a_{z'} = \frac{\|\bar{A}_{z'}\|}{\|A_{z'}\|}$ and $a_z = \frac{\|\bar{A}_z\|}{\|A_z\|}$. By simplifying the performance constraint from Eq.(15) to Eq.(19) and Eq.(20), we can see that no matter C_z or $C_{z'}$ is scheduled in the $(k+1)$th time slot, the time delay $l_i^{(k+1)}$ for $i \notin \{z, z'\}$ will be the same, resulting in the same performance cost $J_i((k+1)s)$ from Eq.(6). As a consequence, to compare $Q_z(k+1)$ with $Q_{z'}(k+1)$, we only need to consider the performance costs of C_z and $C_{z'}$. The local schedulers of C_z and $C_{z'}$ takes $O(n)$ computing complexity to check the following condition D,

$$\text{D} \begin{cases} \frac{ln(\frac{a_{z'}}{a_z})}{s} < \|A_z\|(b_z^{(k)} + 1) - \|A_{z'}\|(z + 1 + b_{z'}^{(k)}), o = 1, \\ \frac{ln(\frac{a_{z'}}{a_z})}{s} < \|A_z\|(b_z^{(k)} + 1) - \|A_{z'}\|(n + b_{z'}^{(k)}), o > 1, \\ (n + b_i^{(k)} + 1)s \leq h_i \text{ for } i = 1, \ldots, n. \end{cases} \tag{21}$$

If condition D is satisfied, the local schedulers in both control applications C_z and $C_{z'}$ can consistently decide C_z should use the network in the $(k+1)$th time slot. Otherwise, they can consistently decide that $C_{z'}$ should use the network. At each time slot, only the local schedulers from the "current control application" C_I and its successor $C_{I'}$ need to check the condition D to make the schedule decisions, while the schedulers of the other control applications do not need.

2) Scheduling the Non-control Applications: The non-control applications in the network can aperiodically activate. Comparing with the control applications, the non-control applications are less safety-critical and do not have hard deadlines. We use admission control in Eq.(12) (as introduced in subsection II-B) to accept the non-control applications at the design time, but the admission control is not sufficient to guarantee the stability of the control applications. Therefore, we design a best-effort scheduling to the non-control applications during the runtime to provide stability guarantees. In the best-effort scheduling, an extra time slot can be assigned to an activated non-control application only if introducing this extra time slot does not jeopardize the stability of any control application. When a non-control application E_i, $i = n+1, \ldots, n+m$ wants to use the network in the $(k+1)$th time slot, the local scheduler of E_i needs to check the stability constraint in Eq.(16). This is because scheduling E_i will turn the $(k+1)$th time slot to be an extra time slot, which increases b_i by 1 for all the control applications and may cause them unstable. If the stability constraint in Eq.(16) satisfies, E_i can be scheduled in the $(k+1)$th time slot. Otherwise, E_i will continue to check the stability constraint in Eq.(16) in the following time slots until it can find a valid one to access the network.

3) Priority-based Arbitration: In NCSs, the control applications do not know when a non-control application activates. Whenever a non-control application E_i activates and decides to use the network in a time slot, it will conflict with a control application C_z or $C_{z'}$. To solve this problem, we use priority-based arbitration in the network. For the network that does not support arbitration mechanism like real-time Ethernet, we install an independent processor in the network to act as an network arbitrator. When the local scheduler in an application decides to use the network, it will send a message to the arbitrator. the arbitrator in the network will forward the one with the highest priority to its destination node and discard the messages from the other local schedulers.

All the control applications have the same priority, which is lower than the priorities of the non-control applications. For the non-control applications, the one with more importance to the system will be given a higher priority. We assign the higher priorities to the non-control applications than the control applications for two reasons. One is that the non-control application

TABLE I: Variables in Local Scheduler

Name	Content	Specification
a	$[\|\|A_1\|\|, \ldots, \|\|A_n\|\|]^T \in \mathbb{R}^n$	store $\|\|A_i\|\|$, $i = 1, \ldots, n$, time-invariant.
$\bar{a}$	$[\|\|\bar{A}_1\|\|, \ldots, \|\|\bar{A}_n\|\|]^T \in \mathbb{R}^n$	store $\|\|\bar{A}_1\|\|$,$i = 1, \ldots, n$, time-invariant.
b	$[b_1, \ldots, b_n]^T \in \mathbb{R}^n$	store b_i, $i = 1, \ldots, n$, time-variant.
h	$h = [h_1, \ldots, h_n]^T \in \mathbb{R}^n$	store h_i, $i = 1, \ldots, n$, time-invariant.
I	$I \in \mathbb{R}$	store index of "current-control application", time-invariant.
r_i	$r_i \in \mathbb{R}$	store priority of the application, time-invariant.
acc	boolean	acc =true the application can access the network, otherwise cannot, time-variant.
i	$I \in \mathbb{R}$	store index of the application, time-invariant.
i'	$i' \in \mathbb{R}$	store index of C_i's successor (control application only), time-invariant.
$'i$	$'i \in \mathbb{R}$	store index of C_i's predecessor (control application only), time-invariant.
$b^{(p)}$	$b^{(p)} \in \mathbb{R}$	store b_i in the previous time slot (control application only), time-variant.

does not jeopardize the stability of the control applications when it decides to use the network by checking Eq.(16). The other reason is that the non-control application has no chance to use the network if higher priorities are given to the control applications, since there is always one application C_I qualified to use the network by checking Condition D in Eq.(21) at each time slot.

C. Message Communication Principles in CSS

We design the message communication principles in CSS to ensure the consistency of the local schedule decisions, even under the transmission failures of messages.

1) Data Structure: We assign the processor memory in each local scheduler to store the variables related to the network scheduling scheme. The specification of the variables is shown in Table I. We use r_i^E and r_i^C to indicate the priority r_i of the non-control application and control application, respectively. We use the lower number to denote higher priority, therefore we have $r_i^E < r_i^C$. Note that $r_i^E \neq r_j^E$ for $i \neq j$, and $r_1^C = r_2^C = \cdots = r_n^C$. All the time-invariant variables as shown in Table I are configured at the design time, and the time-variant variables should be updated online. The variables $\{l, b, I\}$ in each local scheduler are initialized as $\{$ **0**,**0**,1$\}$ at the first time slot with $l \in \mathbb{R}^n$ and $b \in \mathbb{R}^n$.

2) Message Transmission Sequence: Each application in NCSs has two transmission components, namely the source component (SC) and the destination component (DC). In the control application C_i, SC is the sensor and DC is the controller. In the non-control application E_i, SC and DC are defined by the engineers at the design time according to the functionality of each transmission component. Each application chooses one of its transmission components to act as the local scheduler. The local scheduler is responsible to notify both SC and DC in the application to use the network. After receiving the notification from the local scheduler, SC and DC in the application begin to send/receive the messages that contain the data in need. When DC in the application receives a message with data in need from SC, we assume one message transmission finishes and DC broadcasts an acknowledgement message over the network to inform all the applications in NCS about the successful message transmission. There are three types of messages in the network:

- Notification Message msg_i^{ntf}: The notification message is to notify the transmission components of the ith application in NCSs to use the network at the current time slot. It only contains the application index i and priority r_i.
- Normal Message msg_i^{nml}: The normal message is to transmit useful data like the plant's state between the components of the ith application, which contains application index i, the priority r_i and the data in need.
- ACK Message msg_i^{ack}: The ACK message is to indicate all the other applications in the network that the message transmission in the ith application at the current time slot is successful. It only contains the application index i and priority r_i.

The message transmission sequence in each application is summarized by Algorithm 1. We use the abbreviation AP$_i$ to denote the application with index i, and use NA to denote the network arbitrator. The message transmission sequence can finish in one time slot when AP$_i$ uses the network.

Algorithm 1 Message Transmission Sequence in AP$_i$

```
if AP_i (C_i or E_i) can use the network then
    Local scheduler of AP_i sends msg_i^ntf to NA.
    if msg_i^ntf has the smallest r_i in NA then
        NA forwards msg_i^ntf to SC in AP_i.
        NA deletes all msg_j^ntf (j ≠ i in the queue).
    end if
    if SC in AP_i receives msg_i^ntf then
        SC in AP_i sends msg_i^nml to DC in AP_i.
    end if
    if DC in AP_i receives msg_i^nml then
        DC in AP_i broadcast msg_i^ack over the network.
    end if
end if
```

3) Working Principle of Local Scheduler: The local scheduler in each control or non-control application needs to update the variables listed in Table I and check the specific conditions to make schedule decisions. At first, the local scheduler sends the notification message if the application can use the network. The message transmission sequence as shown in Algorithm 1 is executed in the network. Upon receiving the ACK message broadcasted in the network, local scheduler updates the time-variant variables in Table I. After that, the local scheduler checks the condition based on the newly updated variables to decide whether the application can use the network or not in the next time slot. We use Algorithm 2 to demonstrate how the local scheduler in AP$_i$ works. In the algorithm, we use

$AP_i.LS$ to denote the local scheduler of AP_i.

IV. Stability Analysis of NCS under CSS Network Scheduling Scheme

In this section, we provide theoretical proof that CSS can ensure the stability of all control applications C_i $(i = 1, \ldots, n)$ in NCS.

Theorem IV.1. *Given the NCS with n control applications C_i, $i = 1, \ldots, n$ and m non-control applications E_i, $i = n+1, \ldots, n+m$, the upper bound of the network-accessing delay for each control application C_i as h_i, all the control applications are stable under CSS network scheduling scheme when message transmission failure does not occur.*

Proof: In CSS, the time slot length s in the time-triggered network is chosen according to Eq.(13), hence we have

$$ns < \min_{i \in \{1,\ldots,n\}} h_i, \tag{22}$$

Moreover, we give extra time slot to C_i or E_i only if the constraint in Eq.(16) satisfies. If messages can always be transmitted successfully in the network, every control application C_i can access the network at least once in any period with length h_i during the runtime. Since h_i is chosen according to Eq.(11), the stability of C_i can be ensured according to the proof of Theorem 2.15 in [13]. Therefore, CSS can ensure the stability of the control applications in NCS without considering the message transmission failure. ■

If the message transmissions always succeed, from Theorem IV.1 we can see that CSS scheme can ensure the stability of all the control applications even when workload varies due to the existence of the non-control applications. However, when message transmission fails randomly, CSS scheme, as all of the network scheduling algorithms, cannot ensure the stability of C_i since the message transmission failure is unpredictable and can happen frequently (i.e. the extreme case is when all the message transmissions fail). When message transmission failure happens in the network, CSS scheme is a best-effort scheme that provides extra time slots to those control applications with large error due to packet dropout.

Lemma IV.2. *Suppose in an NCS with n control applications C_i, $i = 1, \ldots, n$ and m non-control applications E_i, $i = n+1, \ldots, n+m$, the upper bound of the network-accessing delay for C_i is h_i, and the maximum error e_i of C_i in any interval with length h_i can be strictly bounded by $\beta_i \in [0, \infty)$. Then, CSS scheme can guarantee that for any time $t > t_0 + \left\lfloor \frac{\min_{i \in \{1,\ldots,n\}} h_i}{s} \right\rfloor s$ with t_0 as the beginning of the runtime and s as the time slot length we have*

$$\sum_{i=1}^{n} \|e_i(t)\| \leq \sum_{i=1}^{n} \beta_i, \tag{23}$$

where β_i is set according to Eq.(26).

Proof: From Theorem IV.1 and Theorem II.1, CSS can ensure the exponential stability of C_i. Suppose the equilibrium

Algorithm 2 Working Principle of Local Scheduler in AP_i

```
CurrentSlot = k
if acc = true then
    AP_i transmits message (Algorithm 1)
end if
if AP_i.LS successfully receives msg_j^ack then
    if j > n then
        {j is index of E_j}
        b = b + [1, ..., 1]
    else
        {j is index of C_j}
        if i = j & I = j then
            {CurrentSlot is extra slot given to C_i}
            AP_i.LS updates local variables:
            b^(p) = b_i, b_i = 0, b_q = b_q + 1 for q ≠ i.
        else if i = j & I ≠ j then
            {CurrentSlot given to C_i is not extra slot}
            AP_i.LS updates local variables:
            I = i, b^(p) = b_i, b_i = 0.
        else if i ≠ j & I = j then
            {CurrentSlot is extra slot given to C_j}
            AP_i.LS updates local variables:
            b_j = 0 and b_q = b_q + 1 for q ≠ j.
        else if i ≠ j & I ≠ j then
            {CurrentSlot given to C_j is not extra slot}
            AP_i.LS updates local variables:
            b_j = 0 and I = j.
        end if
    end if
else
    b = b + [1, ..., 1].
end if
if AP_i is control application then
    if i = I or 'i = I then
        AP_i.LS checks condition D in Eq.(21)
        if (Condition D satisfies and i = I)||(Condition D does
        not satisfy and 'i = I) then
            acc = true
        else
            acc = false
        end if
    end if
else
    if AP_i activates now then
        AP_i checks stability constraint in Eq.(16)
        if Eq.(16) satisfies then
            acc = true
        else
            acc = false
        end if
    end if
end if
```

of $x_i(t)$ is $\mathbf{0}$, according to the definition of exponential stability, there exist constant a, b such that

$$\|x_i(t)\| \leq a \|x_i(t_0)\| e^{-bt}, \forall t, t_0 \geq 0. \tag{24}$$

For any period $[t, t+h_i]$, according to Eq.(5) and Eq.(24), the error of C_i can be bounded by

$$\begin{aligned} \|e_i(t)\| &\leq \frac{\|\bar{A}_i\|}{\|A_i\|}(e^{\|A_i\|h_i} - 1)\|x_i(t)\| \\ &\leq a\frac{\|\bar{A}_i\|}{\|A_i\|}(e^{\|A_i\|h_i} - 1)\|x_i(t_0)\| \end{aligned} \tag{25}$$

We set the bound for e_i during the time interval h_i as β_i,

$$\beta_i = a\frac{\|\bar{A}_i\|}{\|A_i\|}(e^{\|A_i\|h_i} - 1)\|x_i(t_0)\|, \tag{26}$$

For any interval that contains $\left\lfloor \frac{\min\limits_{i\in\{1,\ldots,n\}} h_i}{s} \right\rfloor$ time slots, the length of the interval does not exceed $\min\limits_{i\in\{1,\ldots,n\}} h_i$, hence e_i for $i = 1, \ldots, n$ during the interval does not exceed β_i. Therefore, for any $t > t_0 + \left\lfloor \frac{\min\limits_{i\in\{1,\ldots,n\}} h_i}{s} \right\rfloor s$, we have the inequality in Eq.(23). ∎

From Lemma IV.2 we can see CSS scheme can provide bounded error of all C_i. Note that Lemma IV.2 works when there is no message transmission failure happens.

V. Performance Evaluation

In this section, a three servo target tracking system is designed to evaluate the performance of the CSS scheme.

A. Simulation Setup

In the network controlled system, there are three servos designed to track a given trajectory $r(t)$. The trajectory $r(t)$ is a square wave with amplitude 1m and changing period 1sec,

$$r(t) = \begin{cases} 1, & t \in [0, 1) \\ 0, & t \in [1, 2] \end{cases}. \tag{27}$$

In each servo control application, the servo's states are transmitted by the sensor to the controller through the network. The servo control application has the state space function in Eq.(1), and the (A_i, B_i, K_i) as well as the closed-loop poles are given in Table II. Note that the delay bound h_i is not computed from Eq.(11) but given according to the real performance of each control application, since Eq.(11) tends to provide a conservative delay bound that is sufficient for the stability but is usually much smaller than the bound in the real case. Since we want to test the target-tracking performance of the servo control application, we use $y_i(t) = [1\ \ 0]x_i(t)$ as the output of each servo. We define the overall target-tracking error of the three servos as $f(t)$, $f(t) = \sum_{i=1}^{3} |y_i(t) - r(t)|$. The metric for system performance is the average value of $f(t)$ during the whole run time, denoted as average integration error (AIE),

$$\text{AIE} = \frac{1}{\text{Total Runtime}} \int_0^{\text{Total Runtime}} f(t)dt \tag{28}$$

The smaller the AIE, the better the scheduling scheme.

We use the dynamic TDMA plugin in TrueTime simulator [15] to construct CSS to schedule applications in NCSs. The network bandwidth is configured as 250 Kbps. The packet size is 135 bits per packet, and the time slot s is set as 4msec. Besides the three servo target-tracking control applications, there is one non-control application in the NCS, which sends real-time traffic aperiodically to the network. Moreover, the NCS may suffer message transmission failure.

TABLE II: Configuration of the Three Servo Motors

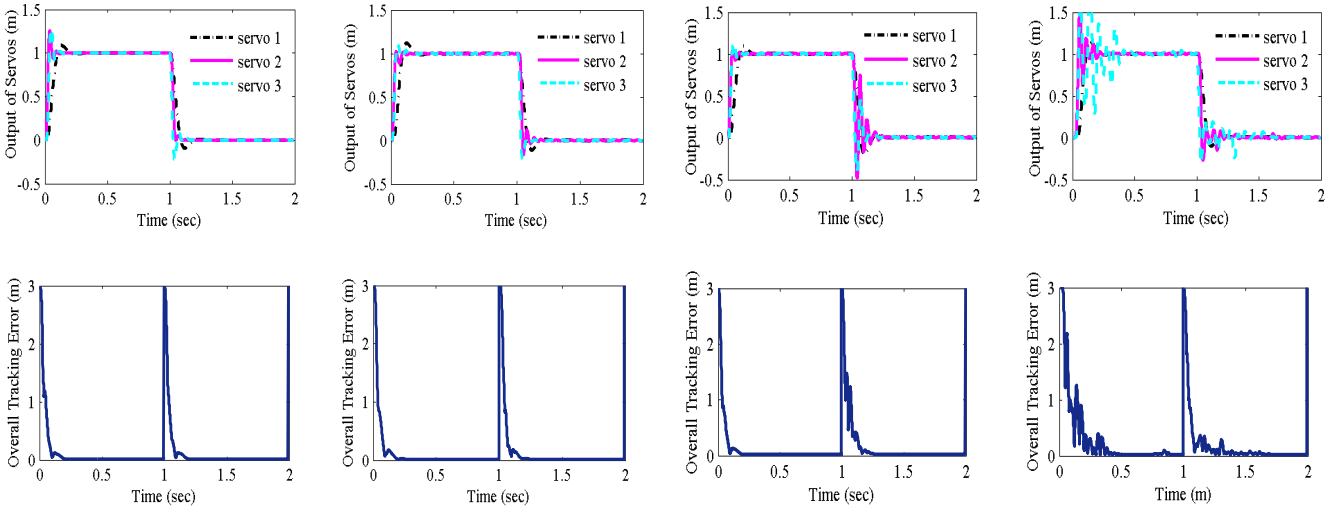
No.	A_i	B_i	K_i	poles	h_i
1	$\begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 1000 \end{bmatrix}$	$[\ 0.80 \quad 0.035\ ]$	$-18 \pm 21.82i$	24
2	$\begin{bmatrix} 0 & 1 \\ 0 & -0.5 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 1500 \end{bmatrix}$	$[\ 1.67 \quad 0.045\ ]$	$-34 \pm 36.66i$	16
3	$\begin{bmatrix} 0 & 1 \\ 0 & -2 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 1200 \end{bmatrix}$	$[\ 3.51 \quad 0.069\ ]$	$-42 \pm 49.08i$	20

B. Performance of CSS under Dynamic Workload

We simulate the performance of CSS under dynamic workload. In our simulation, the probability that the non-control application activates in each time slot, denoted by p, follows a uniform distribution with mean $\mathbb{E}(p) \in [0, 1]$. Therefore, changing $\mathbb{E}(p)$ will change the bandwidth utilization of the non-control application during the runtime, hence change the total bandwidth utilization of the three servo target-tracking control applications. From Table II, we can compute the lower bound of the bandwidth utilization for the three servo target-tracking control applications: $\sum_{i=1}^{m} \underline{u}_i = \frac{4}{24} + \frac{4}{16} + \frac{4}{20} \approx 62\%$ according to Eq.(12). No scheduling policy can work if the non-control application takes more than 38% bandwidth. To test the performance of the three control applications under different bandwidth utilization, we run four times of simulation and configure $\mathbb{E}(p)$ in the four runs as 0.1, 0.2, 0.3 and 0.38 respectively, resulting in the portion of bandwidth available to the control applications as 90%, 80%, 70% and 62% correspondingly. The output of the three servos and the target-tracking error during the runtime in the four runs are shown in Fig. 3. Table III gives the AIE of the NCS in the four runs. With the decrease of the bandwidth utilization of the control applications, the AIE of the NCS increases. The three servos tracks the given trajectory $r(t)$ with high accuracy and low overshoot in the first three runs (the bandwidth utilization is configured as 90% or 80% or 70%). However, when the bandwidth utilization falls to 62%, the target tracking performance of servo2 and servo3 are not ideal, and AIE becomes larger than the values in the first three runs, but CSS can still make the three control applications to be stable. The experiment results shows the effectiveness of CSS for scheduling applications in NCS under dynamic workload.

C. Comparison between CSS and Static Cyclic Scheduling under Message Transmission Failure

NCSs with time-triggered network may suffer message transmission failure. We evaluate the performance of CSS scheme and the static cyclic scheduling algorithm when the message transmission failure happens in the network. The static cyclic scheduling uses static TDMA with cyclic sequence to schedule the control applications, and each control application is assigned only one time slot in a communication cycle. Since the static cyclic scheduling can not handle the dynamic workload from the non-control application, we deactivate the non-control application to compare the performance



(a) 90% (b) 80% (c) 70% (d) 62%

Fig. 3: The Output of Servos and The Overall Target-Tracking Error of CSS under Different Bandwidth Utilization

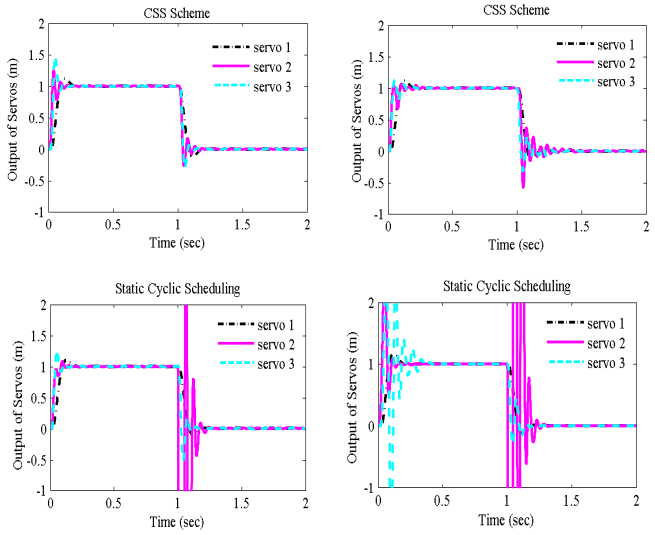
TABLE III: Average Integration Error (AIE) of CSS under Different Bandwidth Utilization of the Control Applications

Bandwidth Utilization	90%	80%	70%	62%
AIE (m)	0.0898	0.0931	0.0969	0.2115

of CSS and the static cyclic scheduling. In our simulation, the probability that the message transmission failure happens in each time slot, denoted by q, follows a uniform distribution with mean $\mathbb{E}(q) \in [0, 1]$. First, we configure $\mathbb{E}(q)$ as 15% and test the performance of CSS and the static cyclic scheduling as shown in Fig.4(a). While CSS can provide a small overall target-tracking error of the three servos, the static cyclic scheduling cannot. The output of servo 2 deviates from the target trajectory at around t=1sec when $r(t)$ suddenly changes from 1m to 0m. To compare the performance of CSS and the static cyclic scheduling under an intensive packet dropout situation, we configure $\mathbb{E}(q)$ as 30%. As shown in Fig. 4(b), CSS can still provide the desired output, but the static cyclic scheduling causes large target-tracking error of servo2 and servo3 during the runtime. This is because servo2 and servo3 are more sensitive to the access delay and message transmission failure, instead of scheduling them according to the fixed sequence, CSS can flexibly add extra time slots in each cycle to them. AIE of the CSS and static cyclic scheduling under different message transmission probability is given in Fig. 5. CSS provides a better system performance with smaller control errors than the static cyclic scheduling when message failure happens in the network.

VI. Related Work

There is a large body of literature on scheduling of real-time traffic within control networks. To be specific, Branicky *et al.* [1] applied the rate monotonic scheduling (RMS) algorithm to schedule a set of control applications in NCS. Ren *et*



(a) 15% (b) 30%

Fig. 4: Performance of CSS and Static Cyclic Scheduling under Different Message Transmission Failure Probability

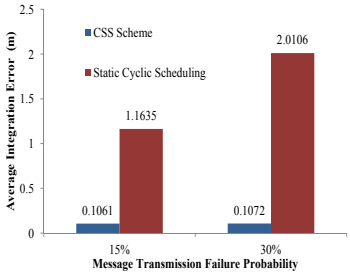


Fig. 5: Average Integration Error (AIE) under Different Message Transmission Failure Probability

al. [16] provided a QoS management scheme for parallel NCSs. Hong [17] proposed a scheduling algorithm to adjust the data sampling time so that the performance requirement of each control loop is satisfied while the utilization of network resource is significantly increased. Later, an extension of this algorithm for the bandwidth allocation applicable to the CAN protocol was provided in [18], which can satisfy the performance requirements of real-time application systems and fully utilize the bandwidth of CAN. Rehbinder and Sanfridson [19] provided an optimal off-line scheduling method by leveraging the control theory. However, these algorithms are static without considering the workload variation.

Many researchers proposed dynamic methods for co-design of network scheduling and control applications. For example, Walsh and Ye [3] presented a dynamic arbitration technique to grant network access to the control loop with the highest error using maximum-error-first with try-once-discard (MEF-TOD). Yepez *et al.* [4] provided a large error first (LEF) scheduling algorithm based on the continuous feedback from the QoC of the control applications. Similar to [4], Xia *et al.* [5] proposed a scheduling algorithm based on the importance of each control application. However, these dynamic scheduling algorithms in [3]–[5] need a centralized scheduler to make the schedule decision and focus on the priority-based network like CAN.

Dynamic TDMA like the conditional static scheduling [6], [20] was proposed to schedule the dynamic workload in embedded systems with time-triggered network. How to apply the dynamic TDMA to schedule the applications in NCSs and provide stability guarantees is not addressed. In this paper, we apply the conditional state-based scheduling to schedule applications NCSs and propose CSS scheme. CSS target at the time-triggered network and use a distributed agreement to schedule multiple applications in NCS.

VII. Conclusion

To handle workload variation in NCS, we provide a novel scheduling scheme, CSS, to dynamically allocate the network resource among all the control and non-control applications. CSS aims at the time-triggered network, and it is a conditional state-based scheduling scheme that can make on-the-fly changes to the pre-defined scheduling sequence. CSS uses the distributed consensus to make the schedule decisions, where each application in NCSs has the local scheduler to decide whether the transmission components in the application can use the network or not in each time slot. The schedule decisions should be made to improve the overall performance of an NCS and at the same time ensure the stability of all the control applications in the network. Algorithm and message communication principles are designed to ensure the consistency of the local schedule decisions. A servo target tracking system is simulated to evaluate the proposed scheduling scheme. Simulation results demonstrate the advantages and applicability of CSS in networked control systems.

Acknowledements.

This work was supported in part by NSERC DG 341823-07, NSERC DG 357121-2008, FQRNT grant 2010-NC-131844, ORF-RE03-045, ORF-RE04-036, ORF-RE04-039, APCPJ 386797-09, CFI Leaders Opportunity Fund 23090, CFI 20314 and CMC, ISOP IS09-06-037, and the industrial partners associated with these projects.

References

[1] M. Branicky, S. Phillips, and W. Zhang, "Scheduling and Feedback Co-Design for Networked Control Systems," in *Proceedings of the 41st IEEE Conference on Decision and Control*, vol. 2, pp. 1211–1217, 2002.

[2] P. Martí, A. Camacho, M. Velasco, and M. El Mongi Ben Gaid, "Runtime Allocation of Optional Control Jobs to a Set of CAN-Based Networked Control Systems," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 503–520, 2010.

[3] G. Walsh and H. Ye, "Scheduling of Networked control Systems," *IEEE Control Systems Magazine*, vol. 21, no. 1, pp. 57–65, 2001.

[4] J. Yépez, P. Martí, and J. Fuertes, "Control Loop Scheduling Paradigm in Distributed Control Systems," in *Proceedings of the 29th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, vol. 2, pp. 1441–1446, 2003.

[5] F. Xia, X. Dai, Z. Wang, and Y. Sun, "Feedback Based Network Scheduling of Networked Control Systems," in *Proceedings of International Conference on Control and Automation(ICCA)*, pp. 1231–1236,2005.

[6] K. Schmidt and E. Schmidt, "Systematic Message Schedule Construction for Time-Triggered CAN," *IEEE Transactions on Vehicular Technology*, vol. 56, no. 6, pp. 3431–3441, nov. 2007.

[7] A. Ghosal, H. Zeng, M. Di Natale, and Y. Ben-Haim, "Computing Robustness of FlexRay Schedules to Uncertainties in Design Parameters," in *Design, Automation Test in Europe Conference Exhibition (DATE'10)*, march 2010, pp. 550–555.

[8] H. Zeng, M. Di Natale, A. Ghosal, and A. Sangiovanni-Vincentelli, "Schedule Optimization of Time-Triggered Systems Communicating Over the FlexRay Static Segment," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 1, pp. 1–17,2011.

[9] S. Fischmeister, R. Trausmuth, and I. Lee, "Hardware Acceleration for Conditional State-Based Communication Scheduling on Real-Time Ethernet," *IEEE Transactions on Industrial Informatics,*, vol. 5, no. 3, pp. 325–337, 2009.

[10] P. Pop, P. Eles, and Z. Peng, "Performance Estimation for Embedded Systems with Data and Control Dependencies," in *Proceedings of the Eighth International Workshop on Hardware/Software Codesign,*, 2000, pp. 62–66.

[11] S. Fischmeister, O. Sokolsky, and I. Lee, "A Verifiable Language for Programming Real-Time Communication Schedules," *IEEE Transactions on Computers,*, vol. 56, no. 11, pp. 1505–1519, 2007.

[12] R. Alur and G. Weiss, "Regular Specifications of Resource Requirements for Embedded Control Software," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2008, pp. 159–168.

[13] W. Zhang, "Stability Analysis of Networked Control Systems," Ph.D. dissertation, Electrical Engineering and Computer Science, Case Western Reserve University. http://dora.cwru.edu/msb/pubs/wxzPHD.pdf, May 2001.

[14] W. Zhang, M. Branicky, and S. Phillips, "Stability of Networked Control Systems," *IEEE Control Systems Magazine*, vol. 21, no. 1, pp. 84–99, 2001.

[15] D. Henriksson, A. Cervin, and K. Årzén, "TrueTime: Simulation of Control Loops under Shared Computer Resources," in *Proceedings of the 15th IFAC World Congress on Automatic Control. Barcelona, Spain*, 2002.

[16] X. Ren, S. Li, Z. Wang, M. Yuan, and Y. Sun, "A QoS Management Scheme for Paralleled Networked Control Systems with CAN Bus," in *Proceedings of the 29th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, vol. 1, pp. 842–847, 2004.

[17] S. Hong, "Scheduling Algorithm of Data Sampling Times in the Integrated Communication and Control Systems," *IEEE Transactions on Control Systems Technology*, vol. 3, no. 2, pp. 225–230, 1995.

[18] S. Hong and W. Kim, "Bandwidth Allocation Scheme in CAN Protocol," *IEEE Proceedings Control Theory and Applications*, vol. 147, no. 1, pp. 37–44, 2000.

[19] H. Rehbinder and M. Sanfridson, "Integration of Off-Line Scheduling and Optimal Control," in *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, vol. 1, pp. 137–143, 2000.

[20] U. Keskin, "Time-Triggered Controller Area Network (TTCAN) Communication Scheduling: A Systematic Approach," Ph.D. dissertation, Middle East Technical University, 2008.