OUTPUT FEEDBACK CONTROL AND OPTIMAL BANDWIDTH ALLOCATION OF NETWORKED CONTROL SYSTEMS

A Dissertation

by

JIAWEI DONG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Won-jong Kim
Committee Members,	Reza Langari
	Bryan Rasmussen
	Xi Zhang
Head of Department,	Andres Polycarpou

May 2013

Major Subject: Mechanical Engineering

ABSTRACT

A networked control system (NCS) is a control system where sensors, actuators, and controllers are interconnected over a communication network. This dissertation presents a framework for modeling, stability analysis, optimal control, and bandwidth allocation of the NCS. A ball magnetic-levitation (maglev) system, four DC motor speed-control systems, and a wireless autonomous robotic wheelchair are employed as test beds to illustrate and verify the theoretical results of this dissertation.

This dissertation first proposes an output feedback method to stabilize and control the NCSs. The random time delays in the controller-to-actuator and sensor-to-controller links are modeled with two time-homogeneous Markov chains while the packet losses are treated with Dirac delta functions. An asymptotic mean-square stability criterion is established to compensate for the network-induced random time delays and packet losses in the NCS. Then, an algorithm to implement the asymptotic mean-square stability criterion is presented. Experimental results illustrate effectiveness of the proposed output feedback method compared to conventional controllers. The proposed output feedback controller could reduce the errors of the NCS by 13% and 30–40% for the cases without and with data packet losses, respectively.

The optimal bandwidth allocation and scheduling of the NCS with nonlinear-programming techniques is also presented in the dissertation. The bandwidth utilization (BU) of each client is defined in terms of its sampling frequency. Two nonlinear approximations, exponential and quadratic approximations, are formulated to describe

the system performance governed by discrete-time integral absolute error (DIAE) versus sampling frequency. The optimal sampling frequencies are obtained by solving the approximations with Karush-Kuhn-Tucker (KKT) conditions. Simulation and experimental results are given to verify the effectiveness of the proposed approximations and the bandwidth allocation and scheduling algorithms. In simulations and experiments, the two approximations could maximize the total BU of the NCS up to about 98% of the total available network bandwidth.

To my parents

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Dr. Won-jong Kim for his time and effort throughout my doctoral study at Texas A&M University. Without his guidance this dissertation would not have been completed.

I wish to thank Drs. Reza Langari, Bryan Rasmussen, and Xi Zhang for serving as my advisory committee members. I sincerely appreciate their valuable guidance and comments for my research.

I would like to thank Min-Hyung Lee and Naveen Kumar Bibinagar for their help, discussions and inspiration to the NCSs research. My thanks also go to Yi-chu Chang and Young Ha Kim for their wonderful friendship in the precision lab. Thanks also go to my friends, colleagues, and department faculty and staff members for making my time at Texas A&M University a great experience.

My special appreciation goes to our research sponsor. This work was supported in part by Texas A&M University Program to Enhance Scholarly and Creative Activities under Grant No. 2010-SAC-8779.

Finally, but most importantly, this dissertation is dedicated to my family for their continued love and support that have enabled me to complete my Ph.D. degree. Without their unconditional love, encouragement, and support, I could have never come so far.

NOMENCLATURE

BU	Bandwidth Utilization
CPU	Central Processing Unit
DIAE	Discrete-Time Integral Absolute Error
EDF	Earliest Deadline First
IP	Internet Protocol
KKT	Karush-Kuhn-Tucker
LAN	Local Area Network
LMI	Linear Matrix Inequality
LQG	Linear Quadratic Gaussian
LQR	Linear Quadratic Regulation
LSM	Least Square Method
MSMC	Multiple-Server-Multiple-Client
NCS	Networked Control System
PWM	Pulse-Width Modulation
RM	Rate Monotonic
SSMC	Single-Server-Multiple-Client
SSSC	Single-Server-Single-Client
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WLAN	Wireless Local Area Network

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Page

Page

Page

LIST OF TABLES

1. INTRODUCTION

Networked control systems (NCSs) arose in the interdisciplinary development of computer networks, communications, sensing technologies, and control theories. Other disciplines as mechatronics and embedded technologies also support the development of the NCSs. High-speed Ethernet and Field-bus successfully improved reliability and stability of the NCSs than ever before and promoted their applications in aerospace, manufacturing, process control, teleoperation, exploration, etc. The study of the NCSs has been an active and attractive research area in the past several years due to their broad applications, such as mobile sensor networks [1], remote surgery [2], haptic collaboration over Internet [3–4], automated highway systems [5], and unmanned aerial vehicles [6]. An NCS can be defined as a hybrid system of sensors, actuators, and controllers, which are distributed and interconnected in locations. Reference inputs, plant outputs, and control inputs of an NCS are exchanged over communication networks. This special structure defines an NCS as a distributed closed-loop real-time feedback control system.

The basic functionalities of an NCS include data acquisition (sensors), control commands (controllers), communication (networks), and actuation (actuators). From a larger scope, research of the NCSs can be categorized into two areas [7]:

(1) Control of networks. The control of networks, from computer science perspective, focuses on the research and study of the communication network

itself. The research of interest includes network protocols, routing controls, congestion controls, etc.

(2) Control over networks. The control over networks, from system and control perspective, focuses on control strategies and controller designs that use the networks as data transmission media. The aim is to reduce the effects from the existence of the networks and to maintain system stability and performance.

As discussed above, the research of the control over networks focuses on control methodologies rather than design of the network protocols and analysis of network behaviors. Our interest in this research is to design and analyze the control algorithms of the NCS so that the system stability and performance can be guaranteed and maintained at a specified level. Hence, this dissertation will mainly discuss control issues of an NCS from the control over networks perspective.

1.1 OVERVIEWS OF CONTROL OVER NETWORKS

The beginning of the research of the control over networks could be traced back to publications of "Integrated communication and control systems" authored by Halevi and Ray in 1988 [8–9]. The authors first combined the control systems and the communication networks and named it as integrated communication and control systems in the papers. Ever since, many research institutes and commercial companies have shown great interests in applying the control over networks to their research or practical applications such as remote industrial controls, factory automations, and other areas. The classification of the control over networks depends on communication architectures

between plants and controllers, which can be roughly categorized into three modes: (1) teleoperation, (2) supervisory control, and (3) NCS.

1.1.1. Teleoperation

Teleoperation pertains to the operation of a machine at a certain distance. By distance, it can refer to a physical distance where operators are separated from to-be-controlled systems, actuators, and sensors, or to a change in scales, for instance, remote microscopic-level surgery [10]. Teleoperation arose in the needs of on-board manipulation systems in dangerous or hazardous environments. It is most commonly associated with robots but can be applied to a whole range of circumstances that include radiation sites, nuclear materials cleanups, underwater inspections, explorations, manufacturing, military applications, etc [10–12].

In traditional teleoperation systems, however, the operators must depend on feedbacks provided by real-time sensory feedback systems to perform subsequent actions as shown in Fig. 1. This is possible when latency in the system is minimal so that the system performance could be guaranteed. An unsatisfactory performance may occur due to the lack of local control mechanisms on the plant sites. Moreover, the operators' limited perception of environments could also result in a poor performance. For these reasons, researchers have been focusing their research attention on supervisory control.

1.1.2. Supervisory Control

The development of supervisory control is based on client-server architectures. The sensors, actuators, and controllers are all located at the controlled plant side as shown in Fig. 2. Compared to a teleoperation system, in the supervisory control,

operators usually give high-level symbolic or analogical instructions to the controlled plant remotely instead of directly controlling the remote manipulation system. Unlike a teleoperation system, a supervisory control system introduces autonomous control loops to the remote site. Hence, the controlled plant can be processed continuously and autonomously. The operators monitor the system performance all the time and modify control algorithms when necessary.

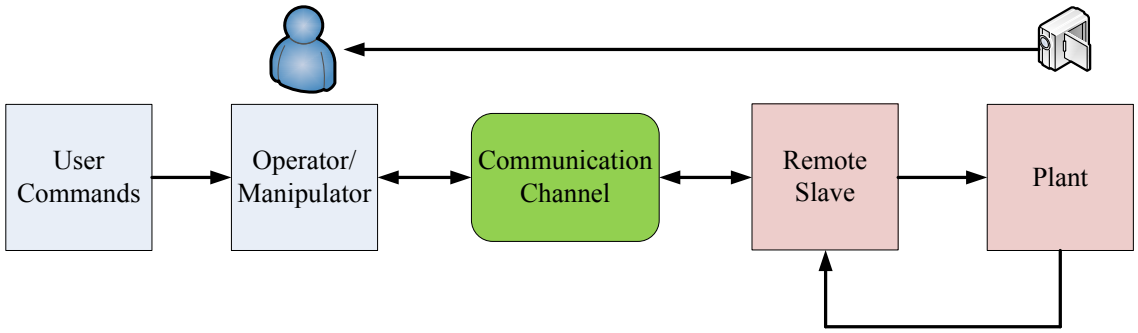


Fig. 1. Block diagram of a teleoperation system

In the last decade, several tele-robots and test beds were established using the Internet as a supervisory control medium [13–18]. The Mercury project was the first successful application of applying the Internet for supervisory control [13]. Luo et al. applied the supervisory control technique to develop a desktop rapid-prototyping system [14]. Garcia et al. developed a tele-robotic system using supervisory control based on a hybrid control approach [15]. Srivatsava designed an Internet-based supervisory control system that operators could monitor process and sent corrective commands to controller from anywhere on the Internet [18].

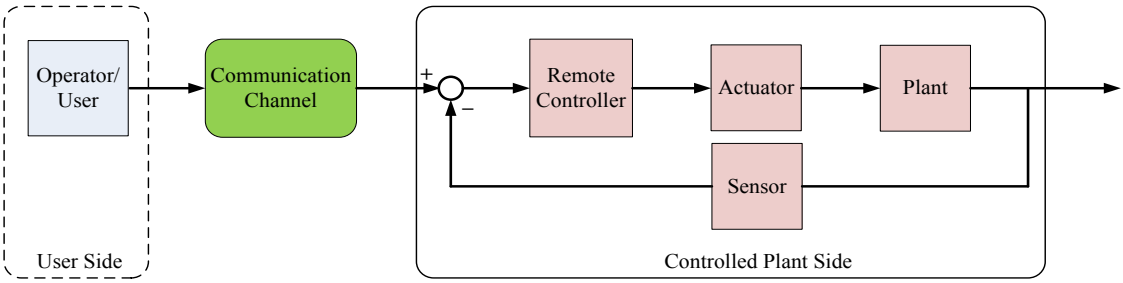


Fig. 2. Block diagram of a supervisory control system

1.1.3. NCS

The communications of Internet-based teleoperation and supervisory control systems are generally unidirectional in nature. The Internet is applied to send instantaneous feedback for monitoring, and operational commands for correcting system actions in emergency. The feedback from the controlled plant is not directly applied to the control-command decisions or the control algorithms modification by the controller itself. However, in the NCS, all nodes, including sensors, actuators, and controllers, are assumed to be interconnected bidirectionally via a communication channel. Furthermore, decisions of the operational control algorithms or commands directly depend on the feedback from the controlled plant side. The control loop is closed over the network with the data packets exchanged on sensor-to-controller link and controller-to-actuator links.

In general, an NCS mainly consists of the controllers, plants, and communication channels. The plants are usually continuous-time systems whereas the controllers are discrete-time systems. The output of the plants are discretized and fed back to the

controllers via the communication channels. The controllers will send control inputs to the plants within the current sampling period, if possible.

In an NCS, the controllers, sensors, and actuators can be distributed at different levels of physical locations [19]. Multiple controllers or multiple plants can exist in the same NCS. The framework of an NCS that includes one controller controlling one client, one controller controlling multiple plants, and several controllers collaboratively controlling one plant, is shown in Fig. 3. Also, there might be other users who do not belong to the NCS share the same network.

For simplicity of the analysis of an NCS, the framework of an NCS in Fig. 3 can be represented in a block diagram shown in Fig. 4. Figure 4 illustrates one of typical structures of an NCS, a direct structure [20]. The NCS with a direct structure is composed of a controller on one side of the communication channel and a remote system containing a plant, sensors, and actuators on the other side. Applications, such as a distance learning lab [21] and DC motor speed-control systems [22], follow this direct-structure framework.

The other structure of an NCS, a hierarchical structure, is shown in Fig. 5. Contrast to the direct structure, it has a remote controller at the remote system side. This remote controller works as a complementary controller to the main controller. In this structure, the main controller generates control inputs to the remote system, and the remote controller executes the control inputs in a local closed-loop manner with possible modifications based on the system feedback in real time. Under situations of large time delays and data packet losses, the remote controller can generate compensational control

inputs to maintain the system performance with the absence of the main controller. Hence, the NCS with the hierarchical structure has a better real-time performance than the one with the direct structure. Applications of the NCS with the hierarchical structure include mobile robots in [23], a modified teleoperation system in [24], etc.

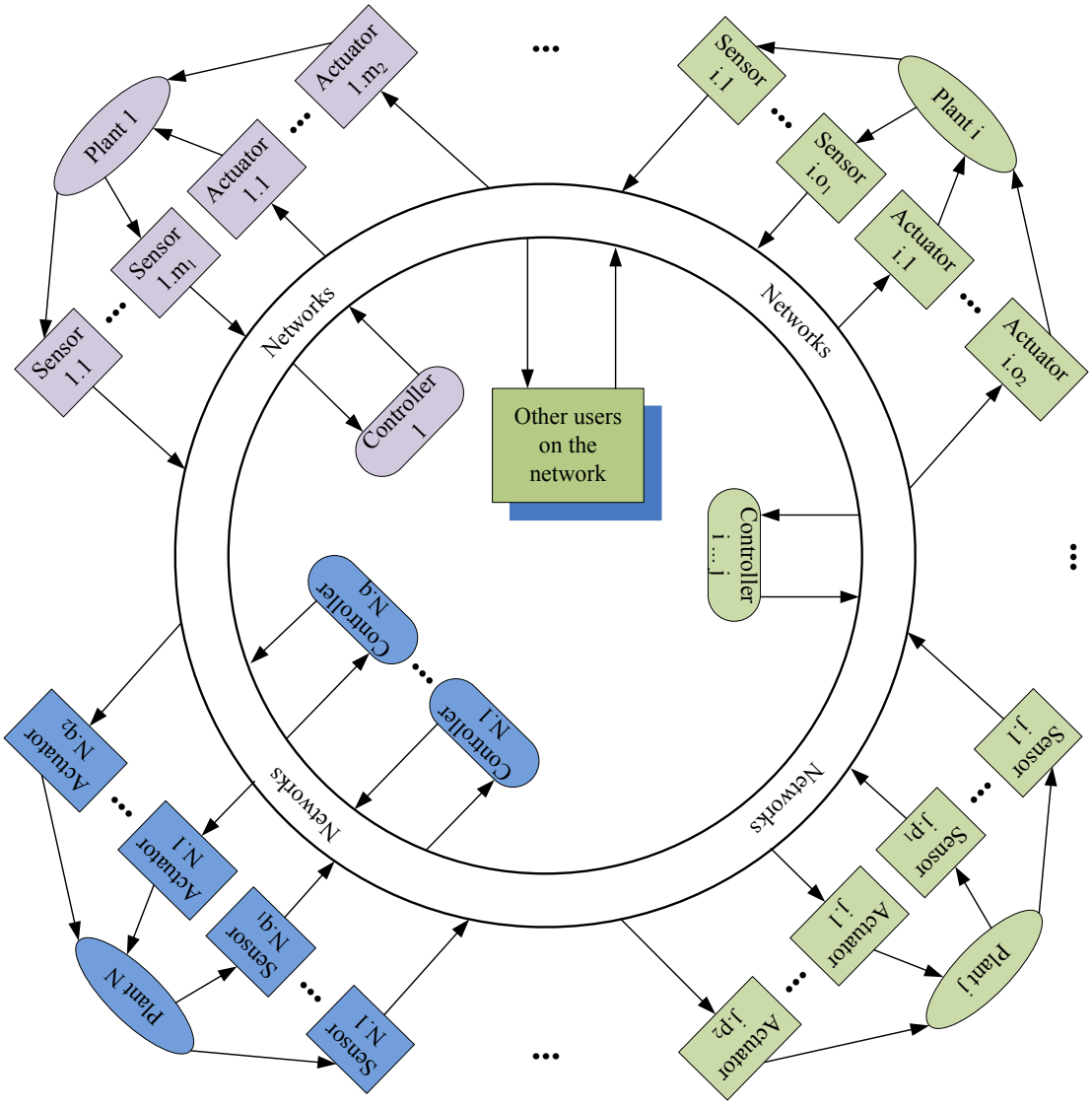


Fig. 3. Representative framework of an NCS

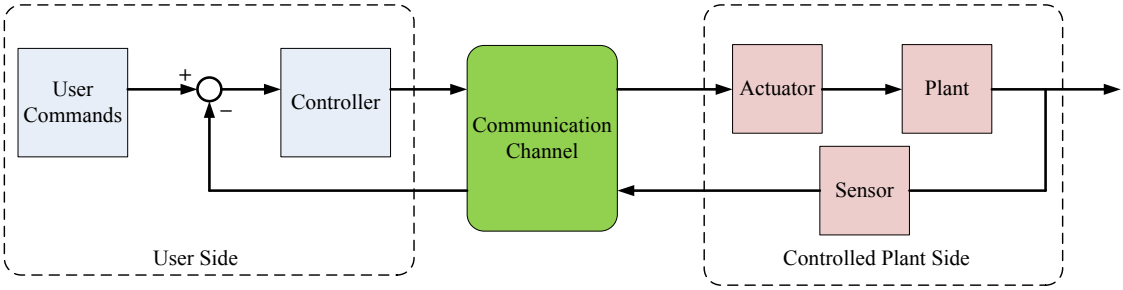


Fig. 4. Block diagram of a typical NCS with a direct structure

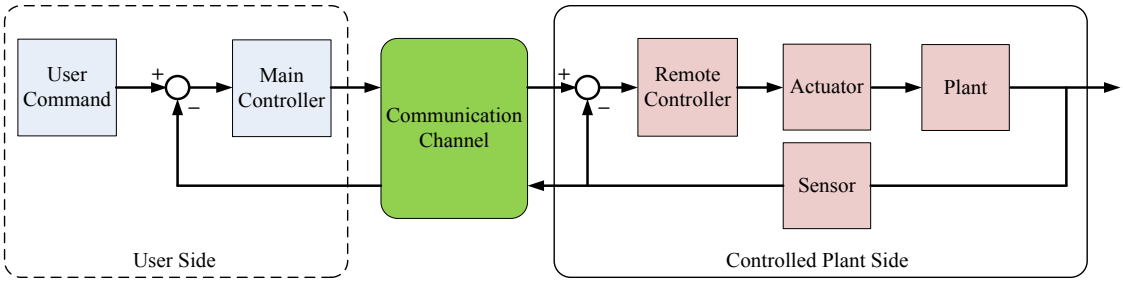


Fig. 5. Block diagram of a typical NCS with a hierarchical structure

This dissertation will focus on one of the most prominent and general forms of the NCS with a direct structure that has a single controller and multiple plants. Several control issues will be discussed and analyzed based on this framework. Hereafter, the nodes containing the controllers will be presented as Server, and the ones containing the sensors, actuators, and plants, Client, respectively. A representative framework of the NCS in this dissertation is given in Fig. 6. In this architecture, all the clients compete for resources, such as bandwidth, central processing unit (CPU) time, or battery to guarantee their stability and system performance. Note that the results in this dissertation can also be applied to the NCS with multiple controllers and multiple plants and the one with the hierarchical structure with appropriate modifications.

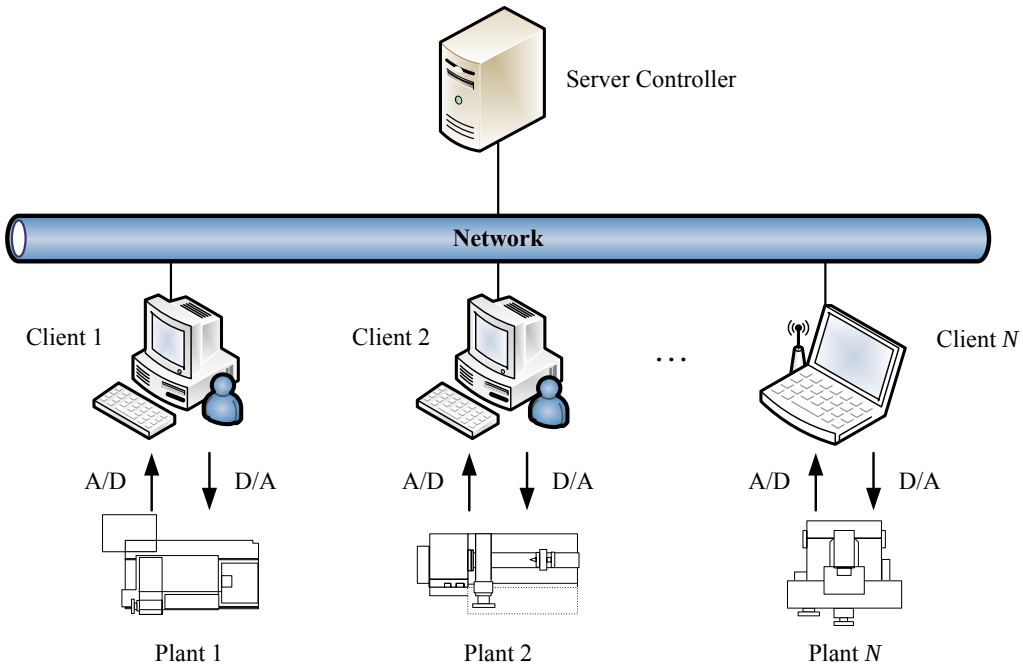


Fig. 6. An NCS with a single server and multiple clients

1.2 INTEGRATED DESIGN ISSUES IN THE NCS

The NCS has advantages of remote operation and control, easy setup and maintenance, increased flexibility and reliability, etc. However, the existence of the networks or other communication channels will inevitably bring more complicated control issues to the NCS such as network-induced time delays, packet losses, optimal sampling periods, resource allocation, network scheduling, etc. Among all the control design issues, the network-induced time delays and packet losses can be generally categorized as the NCS's stability issues, and the optimal sampling periods, resource allocation, and network scheduling can be generally categorized as the NCS's performance issues. Note that these control issues can be coupled and integrated.

1.2.1. Network-Induced Time Delays

Unlike point-to-point connection in traditional control systems, the networks or other communication channels in the NCS will consume longer time on data-packet generation, transmission, processing, etc. For the success of the NCS, either the sensor feedback or the control input must be sampled, encoded, and packed in a data packet, transmitted over the network, and decoded and calculated at the receiver sides. This process significantly outstands the NCS from the traditional control systems. The overall time delays of an NCS can be very stochastic due to the nature of the network communication. In general, the network-induced time delays in the NCS will include

(1) Data-packet-generation delays. Data at each node in the network need to be sampled and capsulated in a single packet or multiple packets before being sent out.

(2) Data-packet-queuing delays. When the network is occupied by other clients or non-NCS users, the data packets will be hold and put in a buffer until the network is available for the next transmission.

(3) Data-packet-transmission delays. The transmission delay is the time consumed by the data packets transmitting over the network. It depends on lengths of communication cables, sizes of the data packets, paths chosen by routers or switches on the network, etc.

(4) Data-packet-processing delays. The processing delay mainly includes the data-packet decoding time after its arrival at the receiver node and its corresponding calculation time.

1.2.2. Network-Induced Packet Losses

The network-induced time delays is not the only control issue brought by the networks, packet losses can be another one. In the traditional control systems, the data are assumed never lost in the transmission. However, the data transmitted over the networks can be lost during the transmission. In an NCS, when the nodes such as sensors, actuators, and controllers exchange the data packets simultaneously over the networks, data-packet collision, network congestion, and connection failure may take place because of the network bandwidth limit and other uncertainties in the network. The possible reasons of the packet losses can be

(1) Physical failures of connections. When communication cables break down for physical reasons, links among each node are disconnected so that the data packets over the network cannot be transmitted to their destination nodes.

(2) Frequent communication congestions. Communication congestions are inevitable due to the share of the links and the network bandwidth limit. When the network is busy or does not have enough bandwidth for current transmission, the data packets are queued in a buffer and wait for a retransmission after a certain time threshold. If retransmissions fail certain times, the data packets will be dropped off by the network protocols. Overflown of the routers or switches on the network can also cause the data packet losses.

(3) Disorder of the data packets. The nodes of the NCS do not designate a data transmission path over the network for each transmission. The paths are chosen by the routers or switches in the next available manner on the network.

Therefore, the later transmitted data packets may arrive first than the earlier transmitted ones. The disorder of the data packets will be considered as packet losses since they are outdated.

(4) Other network uncertainties. Uncertainties such as utilization of the network from non-NCS users or electronic noise may also cause packet losses.

1.2.3. Optimal Sampling Periods

Traditional classical controls assume that the computer control systems have periodical invariant sampling periods. This assumption simplifies the analysis of the control systems. The system performance inversely depends on the sampling period. The smaller the sampling period, the better the system performance. However, the existence of the networks in the NCS complicates this relation between the system performance and the sampling period. A smaller sampling period increases the numbers of the data packets transmitted in the networks, which bring longer time delays overloading the networks, and may destabilize the systems eventually.

A performance chart will provide a clear insight of choosing the optimal sampling periods for an NCS [25]. Figure 7 illustrates a comparison of the system performance versus the sampling period for the continuous-time control, digital control, and NCS. Given a control law, the worst, acceptable, and best sampling periods can be chosen based on control system specifications. The performance axis in Fig. 7 reflects a subset of the control system specifications. Since the performance of the continuous-time control is not a function of the sampling period, it is then a constant for the given control inputs. The performance index only depends on the sampling period without

other uncertainties for digital control. In this case, the performance index is an inverse function of the sampling period in simplicity. The performance degradation point *A* in the digital control can be estimated based on relationship between the control system bandwidth and the sampling periods. For the NCS, point *B* can be determined by investigating characteristics and statistics of the network-induced time delays. As the sampling period gets smaller, the network traffic loads become heavier, the possibility of more contention time delays or packet losses increases, and worse performance results will be exhibited. This causes the existence of point *C* in the NCS.

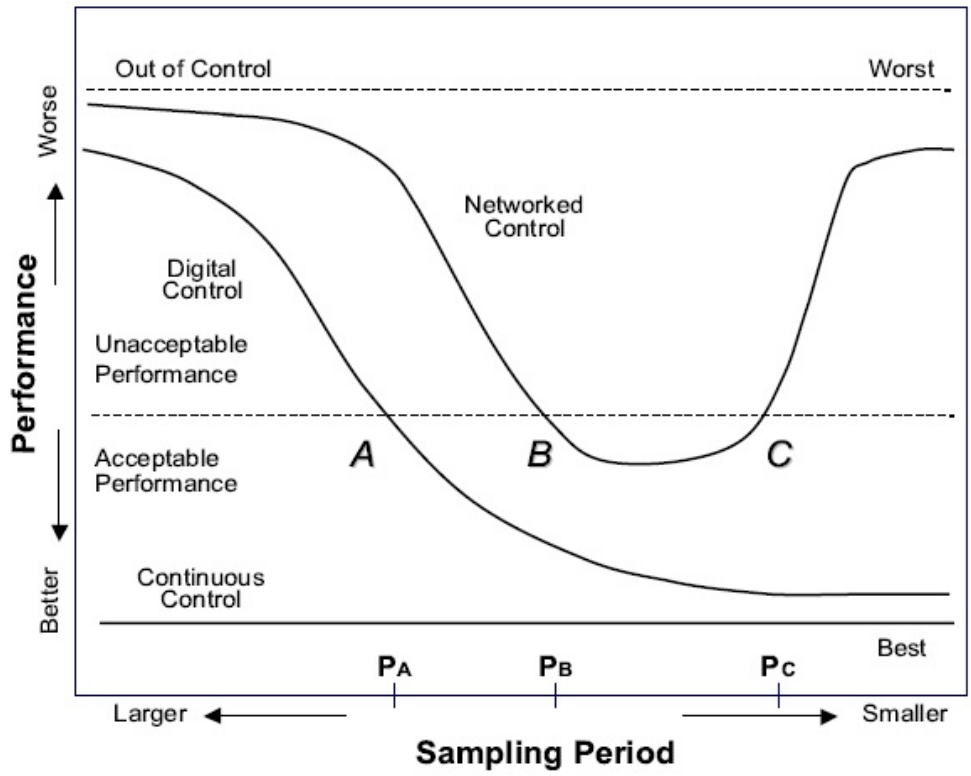


Fig. 7. Performance comparisons of the continuous-time control, digital control, and NCS [25]

From Fig. 7, one can see that the performance index of points *B* and *C* are at the same level. Hence, for an NCS, point *B* should be chosen as it has the same performance as point *C* but introduces less data packets in the network which may reduce the possibilities of longer time delays and save the resource for other clients. Note that the performance of the digital control might be degraded as the sampling period decreases to the hardware limits of the system. Therefore, the sampling period of a digital control system cannot decrease indefinitely. Figure 7 explains the system performance versus the sampling period within the hardware limits of the system.

1.2.4. Bandwidth Allocation and Scheduling

One unique nature of the NCS is that it is a shared network. In general, several clients in an NCS may share one single controller and the network. All these clients compete for the CPU time or network bandwidth to guarantee the stability and system performance. Because of limited computational resources and network bandwidth, necessary resources could not be assigned to each client as required. Fair resource allocation to each client can be critical to the stability and system performance of an NCS. A certain sampling period is necessary to guarantee the stability and system performance of each client. The stability and system performance depends not only on control methodologies but also scheduling of all the data packets in the same link. An optimal scheduling algorithm will not only reduce the time delays but also save more bandwidth for other possible users on the network. To perform a better design of an NCS, both of its control and communication aspects need to be considered [26].

Therefore, a co-design of both controls and network resource allocations must be applied to the NCS design [27–28].

1.3 OBJECTIVES

Considering the discussed control issues and performance evaluation problems of the NCS, design of an NCS can be very challenging and requires not only robust control methodologies that compensate for the time delays and packet losses in the network but also optimal network resources and scheduling algorithms to guarantee the performance of collaborative operation of all the clients. To maintain the stability and even a better system performance of the NCS, our objectives are to propose an effective control method to fully compensate for the effect of the time delays and packet losses in the network and present an optimized bandwidth allocation algorithm to achieve an optimal system performance of the entire NCS. We expect our control methodology can statistically compensate for various levels of the network-induced time delays and packet losses and guarantee the stability of the NCS. Since the NCS contains multiple dynamic systems as the clients that require various sampling periods and controller considerations, we expect our performance evaluations can be fair to each client and eventually to achieve the most optimal system performance of the entire NCS with available network resources based on certain given scheduling algorithms. By fairness, it refers to the bandwidth allocation of each client is based on certain criteria that balance the network bandwidth according to preset priorities or weights. Because of the various system specifications and dynamics, evenly distributed network bandwidth does not guarantee an optimal system performance. The priority and weight of a client in the NCS

can be decided based on its system specification, expected performance, or user defined priorities.

1.4 CONTRIBUTIONS

This research focuses on real-time output feedback control and optimal bandwidth allocation of the NCS. The NCS in this research applies a steel-ball maglev system, a DC motor speed-control system that contains four DC motors, and a wireless autonomous robotic wheelchair as test beds to validate proposed control methodologies and algorithms, and optimal bandwidth allocation. Each client has a unique identification number within its data packets to distinguish from each other. We employ an Ethernet-based local area network (LAN) as the communication network. User datagram protocol (UDP) is applied as the communication protocol in the NCS. In this research, various control issues involved in the NCS are studied. Accomplishments and developed algorithms will be illustrated later in following sections.

Major accomplishments of this research include (1) validation of flexibility and performance of a multiscale wireless/wired NCS that consists of three different types of dynamic systems (fast, medium, and slow clients) with distinct time scales. (2) presentation of an output feedback control methodology based on Markov chain to stabilize and control the NCS. The random time delays in the controller-to-actuator and sensor-to-controller links are modeled with two time-homogeneous Markov chains, while the packet losses are modeled with Dirac delta functions. An asymptotic mean-square stability criterion is established to compensate for the random time delays and packet losses in both the controller-to-actuator and sensor-to-controller links

simultaneously. (3) presentation of bandwidth allocation and scheduling of the NCS to guarantee the system performance of each client in the NCS. The bandwidth allocation algorithm will fairly distribute the network bandwidth to each client based on optimal sampling-frequency assignment under the KKT conditions. The scheduling algorithm will schedule each client in a sequence to maximize the system performance and reduce the idle network bandwidth.

1.5 DISSERTATION ORGANIZATION

This dissertation is organized as follow:

Section 1 provides a brief introduction of the control over networks and their applications. Fundamental structures and current control issues of the NCS are introduced. This section also gives the objectives and contributions of this research.

Section 2 explains basic research issues in the NCS with details based on the discussions raised in Section 1. Existing research results are reviewed and compared regarding to the stability and system performance of the NCS.

Section 3 presents details of fundamental concepts in the NCS such as the time delays, packet losses, bandwidth definition, etc. This section also describes hardware and software setups of the NCS that is used in this research. The experimental assumptions and the architecture of the NCS are illustrated in details. Several analytical results are given in the end.

Section 4 explains a Markov-chain-based output feedback control methodology of the time-delay and packet-loss compensation of the NCS. An asymptotic mean-square stability criterion is established with a Lyapunov approach. This section also presents an

implementing control flow of the proposed output feedback controller. Experimental results are illustrated to verify the effectiveness of the proposed control method.

Section 5 presents exponential and quadratic approximations for the purpose of bandwidth allocation and scheduling of the NCS. A scheduling algorithm is given to achieve the bandwidth allocation methodology that is proposed for experimental verification. Simulation and experiments are conducted to verify the presented approximations and their performance.

Section 6 concludes and summarizes the achievements and contributions of this dissertation. Suggestions for future work and research direction are discussed at the end of the section.

2. LITERATURE REVIEW AND RESEARCH MOTIVATION

In control applications of modern industry, functional nodes such as sensors, actuators, and controllers are geographically distributed. For the success of a distributed control application, all the nodes have to exchange information through communication medium. Although there are great potentials in applications of the NCS, several technical challenges in performing real-time closed-loop control of the NCS should be addressed in advance: (1) networks have inevitable time delays and packet losses that are detrimental to the real-time controls, (2) difficulties in assigning required network bandwidth and other resources to the clients because of the sharing of finite network bandwidth and computational resources, and (3) difficulties in deterministically schedule data packets in the network to avoid congestions and preemption of any data packets.

Success of an NCS relies on the performance of the network, optimal time-delay or packet-loss compensation algorithms, fair resource allocation, collaboration of multiple clients, etc. Variability of the time delays and packet losses make the analysis and design of an NCS difficult. Failure of the resource allocation and scheduling of an NCS will also deteriorate the entire system performance. Therefore, the co-design of the optimal control and resource allocation is necessary to a successful design of the NCS.

2.1 REVIEW OF TIME DELAYS AND PACKET LOSSES

The introduction of a communication network into a control system has brought many advantages, such as no additional dedicated wiring, reduced weight and space requirement, ease of system diagnosis and maintenance, increased system agility, etc. On

the other hand, the communication network inevitably presents more constraints such as random time delays and packet losses that make the analysis and design of the NCS challenging. Unpredictable time delays, packet losses, and sporadic jitters in data transmission are some of the control issues associated with the use of the network as a communication medium. The real-time closed-loop control over the networks should accommodate these uncertainties for satisfactory performances. These random time delays and packet losses can degrade the system performance or even destabilize the system. How to compensate for the random time delays and packet losses has become one of the active research areas of the NCS.

Random time delays can be divided into three major categories, time delays shorter than one sampling period, time delays longer than one sampling period but finite, and infinite time delays which can also be considered as packet losses. The analysis and modeling of random time delays can be performed with a deterministic model or a stochastic model.

Zhang et al. analyzed several fundamental issues of the network-induced time delays in the NCS in [29]. The time delays were assumed to be deterministic, and the controller gain was given as a constant. The relationship between the sampling frequency and the time delays was captured using a stability region plot. Methods to compensate for network-induced time delays using a time-domain solution of a plant model were discussed, and experimental results over a physical network were presented. In [30], the NCS was modeled as a switched system, and the controller gain was also set to be a constant as in [29]. Lin et al. discussed stability and disturbance attenuation

issues for the NCS with random time delays and packet losses. The NCS was considered a discrete-time switched system, and then the stability and performance of the NCS could be reduced to corresponding problems for the switched systems. The random time delays were modeled with Markov chains, and the analysis mainly focused on the delays shorter than one sampling period in [31–32]. The control inputs were derived by setting up cost functions of linear-quadratic regulation (LQR) and linear-quadratic Gaussian (LQG) problems. With the proposed methods, Nilsson analyzed distributed real-time control systems and designed controllers taking into account timing behaviors of the network [31]. The results in [31] had been expanded to the case with the time delays longer than one sampling period as in [33]. Hu and Zhu considered two cases of system with either full-state information or partial-state information. The controllers were shown to render corresponding the NCS exponentially mean-square stable. An optimal estimator of the system state was also presented when the system had partial-state information and time delays longer than one sampling period.

In [29–33], the plants were modeled in continuous-time domain. But in practical NCS applications, the systems more or less involve discrete-time domain specifications. These research results from the continuous-time domain could not be directly transplanted to the discrete-time domain to guarantee the stability and system performance. Therefore, control methodologies of the NCS in the discrete-time domain have also been active research areas in the NCS.

Xiao et al. proposed two types of controller-design methods for the NCS in [34]. The authors presented a V-K iteration algorithm to design stabilizing controllers for

specially structured discrete-time jump linear systems, which were used to model control systems with bounded random time delays in feedback loops. Zhang et al. proposed an output feedback method to analyze the time delays of an NCS and assumed the random time delays could only take integer values [35]. Necessary and sufficient conditions of stochastic stability for the systems were obtained in terms of a set of Linear Matrix Inequality (LMI) with matrix inversion constraints. Shi et al. also proposed an output feedback controller design method for the NCS with random time delays [36]. Conditions of stochastic stability were derived in form of a set of LMIs with nonconvex constraints. The product reduction algorithm was employed to obtain two-mode-dependent output feedback controller. In [37], robust control problem of the NCS with norm-bounded uncertainties was studied. A stochastic stability analysis was addressed, and H_2 and H_∞ norms for this system were defined. The H_2 and mixed H_2 / H_∞ control problems were solved in form of a set of LMIs with nonconvex constraints.

In [38], Hu et al. discussed the stabilization problem of the discrete-time NCS with partly known time delays. A delay-distribution-dependent criterion for the mean-square stability of the NCS was derived by using a Lyapunov-Krasovskii functional approach and LMI technique. Yang et al. studied the NCS with unreliable data communications in [39]. An observer-based controller was designed to exponentially stabilize the NCS in the sense of mean square and also achieved the prescribed H_∞ disturbance attenuation level. An estimation method was introduced to compensate for the lost data of the NCS in [40]. The controller design was considered for both the

available and unavailable states, respectively. Some sufficient conditions were derived so that the closed-loop systems were exponentially mean-square stable.

In [41], an integral control of the NCS with the random time delays was studied. The necessary and sufficient conditions were found for zero-state mean-square exponential stability of the NCS. Ye et al. also modeled the time delays and packet losses in the NCS with Markov chains in [42]. Without the augmented state method, however, the computation effort was reduced. The mode-dependent controller for the closed-loop NCS was presented in a LMI formulation via Schur complement theory. In [43], Xiong and Lam proposed two types of packet-loss models—the arbitrary model and the Markov-chain model. The stability conditions of the NCS with the packet losses were given based on a Lyapunov approach. Liu et al. proposed a time-delay-compensation technique using modified model predictive control method [44]. The packet losses were compensated for with predicative packets generated from the same model. The fixed and random time delays were both studied in the research. In [45], Schenato proposed an optimal estimation design for the NCS. The stability of these estimators depended only on an overall packet-loss probability. The algorithms to compute the packet-loss probability and estimator in terms of the error covariance were given as well.

2.2 REVIEW OF BANDWIDTH ALLOCATION AND SCHEDULING

Traditionally, a control design problem is decoupled from software design and implementation considerations. This separation allows control and computer communities to focus on specific problems. Controller designers disregard characteristics of the implementation, computational, and communication resources, but

focus on controller itself. On the other hand, real-time operating system (RTOS) designers take control loop as a periodic task with hard deadlines [46]. In an NCS, however, these two fields are correlated in a closer way so that their separation will lead to poor system performances. Therefore, co-design of the controllers, resource allocation, and control task scheduling is necessary to the design of an NCS.

Al-Hammouri et al. proposed a bandwidth allocation scheme for the NCS in [47]. The authors formulated the bandwidth allocation of an NCS as a convex optimization problem. While ensured stability of each client in the NCS, the scheme allocated the bandwidth in a manner of maximizing the aggregate performance of the entire NCS. Velasco et al. presented a dynamic control approach to achieve the bandwidth management which allowed control loops to consume network bandwidth according to the dynamics of controlled process while attempting to optimize overall NCS performance [48]. Wong and Brockett investigated a state-estimation problem involving finite communication capacity constraints in [49]. A concept of a finitely recursive coder-estimator sequence was introduced. In [50], Wong and Brockett further introduced the concept of containability to tackle problems of stabilization of an NCS through limited-capacity communication networks.

Martí et al. applied a feedback-based method to allocate resources to controllers as a function of current states of the NCS in [51]. Experimental results showed that the scheme increased and maximized the control performance, saved the resources when perturbations occurred, and incurred negligible overheads. Castané et al. applied a feedback scheduler to determine optimal periods of the plants controlled by arbitrary

control laws in [52]. The resource management was shown as an optimization problem, where objective functions related the sampling periods to transient responses of the controlled plants. Belzarena et al. studied the network bandwidth allocation with time reservations in [53]. The situation involved fully distributed solutions over an arbitrary network topology. The allocation was in a given auction reserved for the entire duration of network connection.

One of the very first papers discussed controllers and schedulers is [54]. Seto et al. considered an optimal sampling period selection for a set of controllers of an NCS. A cost function, approximated with an exponential function of the sampling periods, was used to measure the performance of each client in the system. Park et al. presented a scheduling method for the NCS with three types of data—periodic data, sporadic data, and messages (non-real-time asynchronous data such as system broadcasting messages) in [55]. The maximum allowable time-delay bound was used as a basic parameter for the scheduling method, which guaranteed stability of the NCS and was derived from characteristics of given plants. Branicky et al. proposed a co-design approach treating communication protocols and interacting controlled systems as a coupled system in [56]. The communication issues such as network bandwidth, quantization, survivability, reliability, and time delays were considered simultaneously with the control issues such as stability, performance, fault tolerance, and adaptability.

Walsh and Ye studied scheduling of the NCS in [57]. Performance gains were demonstrated by dispensing with queues and dynamically scheduling network traffic. Error bounds of a static scheduler and a dynamic scheduler were defined for stability

analysis of the NCS. In [58], an approach was proposed to implement dynamic scheduling policy for network bus with performance guaranteed. Weiss et al. presented an automata-based scheduler automatically generated from a model of controlled plant and controller. The proposed method allowed adjustments to dynamic conditions such as varying disturbances and network load besides the ensured performance. In [59], Seto et al. considered optimal sampling period selection for a set of controllers. The system performance was approximated by an exponential function in terms of the sampling frequency. The optimal sampling frequencies were calculated from the KKT conditions with convex constraints. Kim et al. proposed a scheduling method to obtain a maximum allowable delay bound for a scheduling of the NCS [60]. The proposed method was formulated in terms of LMI and could yield an improved delay bound. The presented method could handle periodic data, sporadic data, and non-real-time data. In [61], the schedulability of real-time data was defined, and scheduling algorithms were proposed for efficient transmission of a real-time mixed traffic. Simulation showed enhancements in the average network utilization and packet-loss rate for the real-time data.

2.3 MOTIVATION

As discussed above, some researchers [29–32] modeled the time delays as constant parameters, which could not reveal the stochastic nature of the network-induced time delays of an NCS. Other methods [34–40] treated the time delays as random variables governed by a Markov chain or other probability functions. The authors assumed that the Markov-chain model of the time delays could intuitively include the packet losses as well. However, the packet losses actually change the structure of the

models. Hence, the Markov-chain-based packet-loss model assumes that the packet-loss information can be included by the same probability transition from the time-delay perspective will not closely reflect the nature of the NCS.

Meanwhile, various cost functions or performance index functions were applied as objective functions of the network bandwidth allocation problems with certain constraint conditions. However, these methods did not explicitly consider the effects from the network-induced time delays and packet losses brought by frequent data-packet transmissions or higher sampling frequency of a client. Elimination of these time delays and packet losses in the cost functions or performance index functions could possibly dispel the network effects on the system performance in the perspective of the NCS.

Considering possible limitations of previous modeling methods of the time delays and packet losses, and resource allocation and scheduling algorithms, this research aims to propose an real-time feedback control, optimal resource allocation and scheduling algorithm to fulfill control specifications and design goals of an NCS so that the time delays and packet losses can be modeled and compensated for faithfully and the time delays and packet losses brought by frequent data-packet transmissions can be treated as an essential parameter of the system performance index functions. Developed algorithms will be illustrated and experimentally verified in Sections 4 and 5 of this dissertation.

3. KEY ELEMENTS, EXPERIMENTAL SETUPS, AND ANALYTICAL RESULTS OF THE NCS

Increment in geographical distribution of resources requires modern industries to integrate communications and different aspects of control-system design into various levels of industry operations. These distributed resources need to be accessed and controlled through a communication network. This special system architecture with distributed sensors, actuators and controllers via a communication network has caught the interests of many universities and industries. These communication networks with advanced capabilities for reliability and superior performance enable industrial process controls to make use of concept of distributed real-time controls across a large geographic distance. Figure 8 illustrates a fundamental structure of the distributed systems. Users, servers, and clients can all be geographically distributed via appropriate communication media but have full access to each end of the communication media. The users cannot only send commands to either the servers or the clients to control clients' behaviors but also monitor system responses in real time. The servers and clients exchange sensor measurements and control inputs via the network to maintain system stability and performance.

For the research in this dissertation, we assume that the users and servers are combined together under the structure in Fig. 8. Therefore, the NCS hereafter will only contain the servers and clients in its framework.

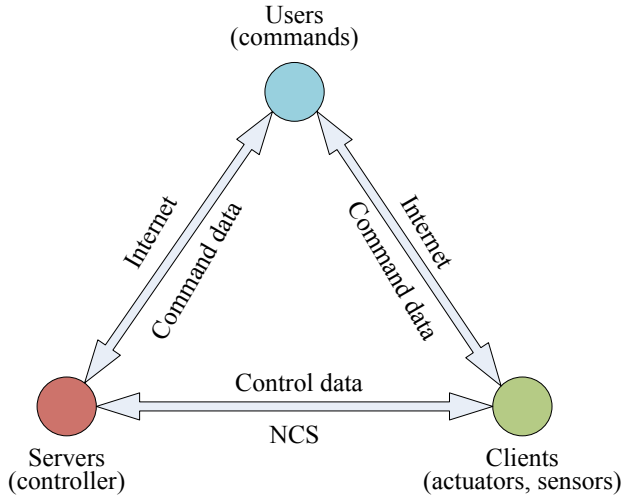


Fig. 8. Internet-based teleoperation system

As discussed in Section 2 and illustrated in Fig. 3, an NCS can include several scenarios that can be categorized by the number of servers and clients involved in the entire framework as follows,

(1) Single-server-single-client (SSSC) framework. This is the simplest scenario of an NCS. On one end of the network is a single server, and the other end, a single client. This scenario requires time-delay and packet-loss compensation but no resource-allocation mechanisms since all the network bandwidth and CPU time will be assigned to this single client.

(2) Single-server-multiple-client (SSMC) framework. This is a scenario of an NCS that has a single server on one end of the network and multiple clients on the other end. This scenario could also be phrased as a collaborative NCS. It refers to a system that needs cooperation of multiple clients which involves the resource-allocation mechanisms. The network bandwidth and CPU time shall be balanced

in a fair manner to guarantee each client's stability and performance as well as schedulability of the entire NCS.

(3) Multiple-server-multiple-client (MSMC) framework. This scenario refers to the most complicated application of an NCS. To successfully achieve the control requirements, multiple servers need to cooperatively manipulate multiple clients in the framework. This framework requires more robust-control algorithms, and resource-allocation and scheduling mechanisms. The resource allocation and scheduling shall be dynamically decided in real time as how to assign servers' resources interchangeably.

In this research, we will focus on the SSMC framework. Regardless of the applications domain of the NCS, this SSMC framework raises several fundamental issues, such as collaborative control and interactivity, time-delay and packet-loss compensation, resource allocation, scheduling, etc. Among all these issues, time-delay and packet-loss compensation of the NCS are some of crucial factors that affect stability of each client in the system. Resources allocation and scheduling are the factors that affect performance of each client and further the entire NCS. Especially for the SSMC framework, the clients will suffer from different levels of time delays or packet losses due to their variant geographical distances to the server, the network conditions, the numbers of clients in the NCS, and the resource assigned to each client. With control issues mentioned here, analysis and modeling of an NCS with the SSMC framework could be difficult to be implemented. To simplify the analysis and modeling procedures,

following assumptions will be made throughout the dissertation without loss of generalities.

(1) Quantization errors in data-packet generation and transmission will be neglected.

(2) All the nodes in the NCS have the same clock resolutions.

(3) Network conditions of all the clients are at the same level regardless of the geographical distances and their system specifications.

(4) All the sensor measurements and control inputs are sent within one single data packet during each control iteration.

(5) The sensor measurements are strict clock-driven tasks, and the control inputs and actuator updates, strict event-driven tasks.

The NCS in this dissertation includes a steel-ball maglev system, a DC motor speed-control system that contains four identical DC motors, and an autonomous wireless wheelchair robot as test beds as shown in Fig. 9. Each client has a unique identification number within their data packet to be distinguished from each other. We employ an Ethernet-based local area network (LAN) as the communication network. User datagram protocol (UDP) and socket programming is applied as communication carrier in the NCS. The wired and wireless TAMULink are chosen to be the data-exchange media at Texas A&M University. The wired TAMULink is the LAN with IEEE 802.3 standard, and the wireless TAMULink is the wireless LAN (WLAN) with IEEE 802.1x standard.

As shown in Fig 9, Clients 1, 2, 4, 5, and 6 represent the wired clients in the NCS that are connected to Server via a LAN. Client 3 represents the wireless client that

includes the wireless robotic wheelchair and a laptop that sends and receives data packets over the wireless network. Since the laptop of Client 3 runs Windows XP operating system (OS), which cannot communicate with Linux OS directly, an Interoperability Suite including a computer operated as an intermediary is set up.

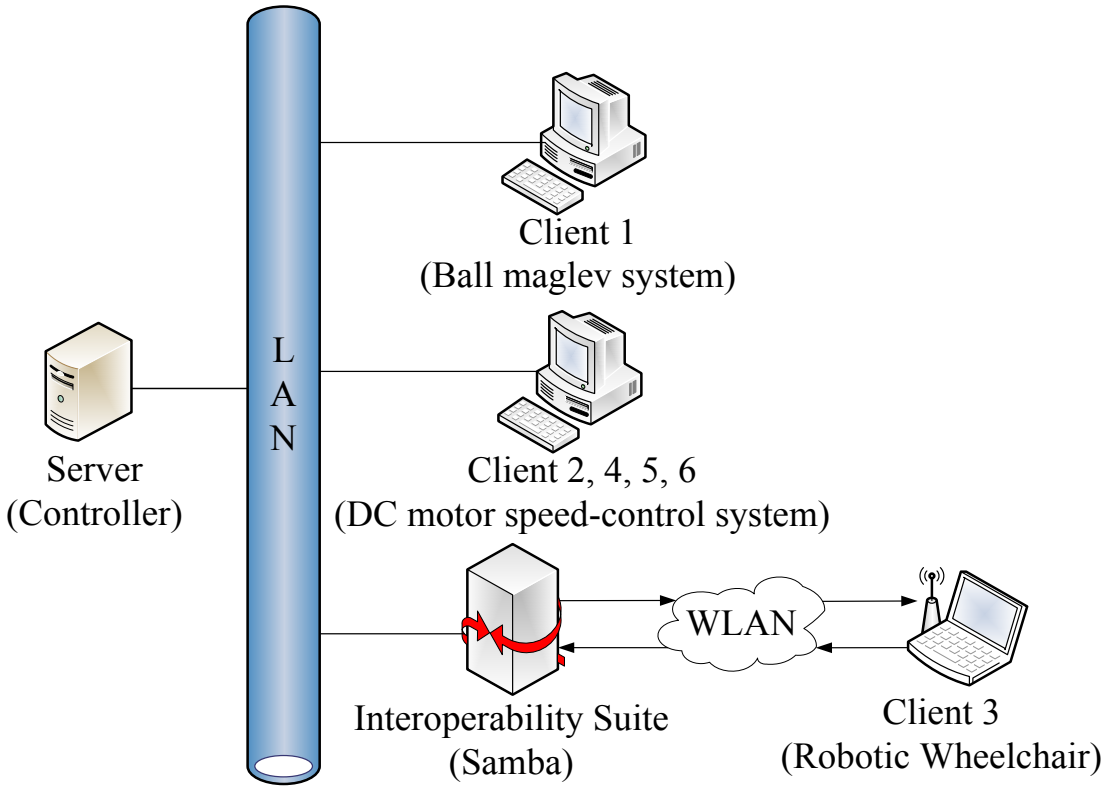


Fig. 9. The NCS architecture with three types of clients

3.1 KEY ELEMENTS

As discussed earlier, the network brings more complex dynamics into the NCS. Time delays and packet losses are the direct effects introduced by the network. As a

matter of fact, these two facts play crucial roles during the design of an NCS. Meanwhile, the structures of the NCS, SSMC and MSMC, require extra control efforts on the resource allocation and scheduling to maintain stability and desired system performance of an NCS.

3.1.1. Time Delays and Packet Losses

To better understand the structure of the network-induced time delays and packet losses of the NCS, consider several control iterations as shown in Fig. 10, whereas the lines with an arrow indicate successful transmission, and the lines with a dot, the packet losses. The red color represents the sensor-feedback data packet from Client to Server, and the green color, the control-input data packet from Server to Client. Table 1 gives nomenclatures of the timing components in Fig. 10.

Figure 10 also illustrates details of the communication in the NCS. In the beginning of experiments, Server waits for the data packets from either Client or Interoperability Suite after Windows sockets setup and UDP connection actives. Client collects sensor measurements from the controlled plant and encapsulates the data segment with necessary headers into one single packet that is ready for transmission. The data packet is then transmitted to its destination if the network is idle or be held in a queue if the network is busy. If no packet losses take place, the data packet will be transmitted through the network to its destination node with a certain amount of propagation delays. The destination node will decode the data packet and implement corresponding calculations. This process achieves the data transmission and calculation from Client to Server. The other transmissions in Fig. 10 follow the same steps.

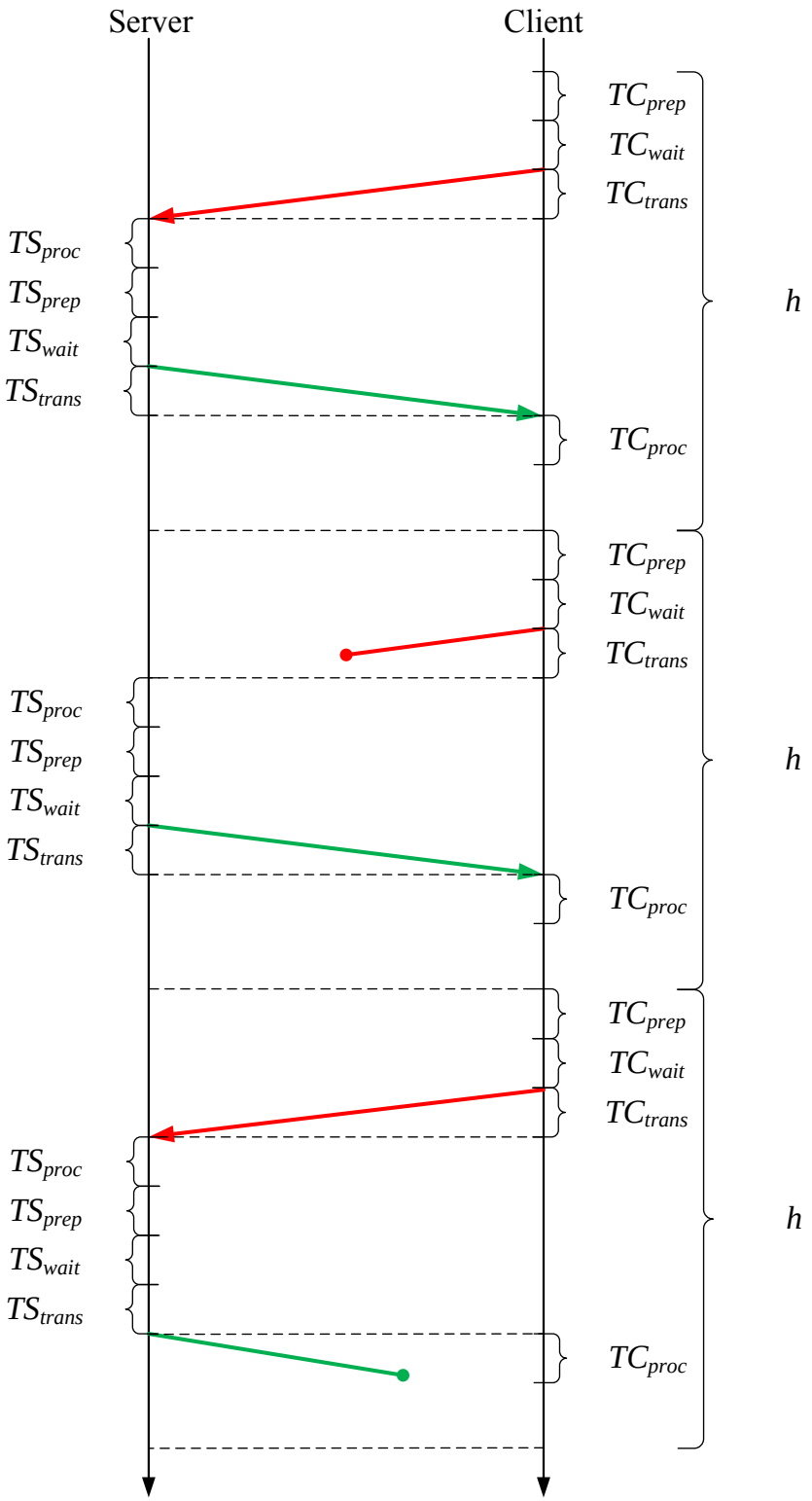


Fig. 10. Time-delay components of the network in several periodic control iterations

Table 1. Nomenclatures of the timing components

Symbol	Description
$T_{*\text{prep}}$	Time taken by Client, Server, or Interoperability Suite to prepare a requested message.
$T_{*\text{wait}}$	Time taken by Client, Server, or Interoperability Suite to wait for network access.
$T_{*\text{trans}}$	Transmission time of a data packet from Client, Server, or Interoperability Suite to its destination node.
$T_{*\text{proc}}$	Time taken by Client Server, or Interoperability Suite to process a data packet.
h	One sampling period of Client.

* can represent *C* (Client), *S* (Server), or *IS* (Interoperability Suite) depend on the content.

In Fig. 10, the first control iteration with a sampling period *h* indicates a general case of the network-induced time delays. The second and third control iterations in Fig. 10 indicate two possible cases of the packet losses in an NCS. The second control iteration shows that the data packet from Client to Server is lost. Consequently, Client will not receive an updated control input from Server because of Server is performed in event-driven based. The details of event-driven based server will be given in Section 3.1.3. The third control iteration in Fig. 10 shows the data packet from Server to Client is lost so that Client will be unable to actuate the plant with the updated control input. Therefore, whenever a data packet is lost in either the client-to-server link or the server-to-client link, Client will be unable to update itself with the latest control input. Hereafter, τ^{ca} and τ^{sc} represents the random time delays in the controller-to-actuator

and sensor-to-controller links, respectively. δ^{ca} and δ^{sc} represents the packet losses in the controller-to-actuator and sensor-to-controller links, respectively.

From Fig. 10, the total time delay τ in one control iteration is given by

$$\tau = TC_{\text{prep}} + TC_{\text{wait}} + TC_{\text{trans}} + TS_{\text{proc}} + TS_{\text{prep}} + TS_{\text{wait}} + TS_{\text{trans}} + TC_{\text{proc}}. \quad (1)$$

In Eq. (1), the preparation time, waiting time, and transmission time are introduced by the network. The processing time is the time interval for Client or Server to process all the data packets. Compared to the traditional controls, the preparation time, waiting time, and transmission time can be classified as the network-induced time delays of an NCS.

The time components in Eq. (1) are difficult to be measured in practice. However, it is possible to measure the time from Server to Client and vice versa by applying a timestamp in the data packet. Therefore, Eq. (2) gives a simple way to measure τ in each control iteration

$$\tau = \tau^{sc} + \tau^{ca} + \tau^{p}. \quad (2)$$

τ^{ca} includes TC_{prep}, TC_{wait}, and TC_{trans}. τ^{sc} includes TS_{prep}, TS_{wait}, and TS_{trans}. τ^{p} is the processing time that includes TS_{proc} and TC_{proc}. Note that τ can be random with respect to the control iterations due to the stochastic nature of the network.

3.1.2. Bandwidth Allocation and Scheduling

The NCS is expected to provide more functionalities and better performance with available resources, such as network bandwidth, CPU time, and batteries. As discussed earlier in this section, this research focuses on the SSMC framework of an NCS so that

all the clients compete for the CPU time or network bandwidth to guarantee their stability and desired system performance. Because of the limits of the computational resources and network bandwidth, sufficient resources could not be assigned to each client as required. Fair resources allocation can be critical to the stability and system performance of an NCS.

Network Bandwidth Definition

From [26], relation between the sampling periods and BUs can be indicated as

$$b_i^k = \frac{\tau_i^k}{h_i^k}, \tag{3}$$

where b_i^k is the BU, h_i^k is the sampling period, and τ_i^k is the total time delay defined in Eq. (2). The subscript i indicates index of the clients in the NCS, and the superscript k indicates the control iterations. Then the BU b_i^k represents a portion of the network bandwidth assigned to Client i at the control iteration k. From Eq. (3), given a certain amount of time delays, a small BU implies a large sampling period and more bandwidth available for other functionalities and control purposes in the same network. If the BU approaches the network bandwidth saturation threshold, the network will be overloaded and induce more time delays or packet losses.

The BU definition in Eq. (3) is associated with and unique for the NCS. It is similar to the execution utilization defined in an RTOS for the purpose of schedulability test, but not so much as the network bandwidth defined from a computer-science perspective.

From the perspective of RTOS scheduling, the task execution utilization or CPU utilization is defined as [62]

$$e_i = \frac{c_i}{h_i}, \tag{4}$$

where e_i is the execution utilization, c_i is the task execution time, and h_i is the period of tasks. The tasks are usually periodic control or calculation tasks of the RTOS. If summation of the execution utilization of each task does not exceed schedulability U of corresponding implemented algorithms, so that $\sum_{i=1}^{N} \frac{c_i}{h_i} \le U$, then hard-time deadline of the tasks can be guaranteed and all the tasks can be scheduled.

Note that, the BU definition of an NCS contains not only the processing time on processors, but more importantly the propagation delays introduced by the networks. However, the execution utilization from the RTOS perspective only considers the processing time on the processors. According to Eqs. (3) and (4), one can see that the RTOS execution utilization is smaller than the NCS BU. The difference is given by

$$\Delta = \sum_{i=1}^{N} \frac{\tau_i}{h_i} - \sum_{i=1}^{N} \frac{c_i}{h_i} = \sum_{i=1}^{N} \frac{\tau_i^{sc} + \tau_i^{ca} + \tau_i^{p}}{h_i} - \sum_{i=1}^{N} \frac{\tau_i^{p}}{h_i} = \sum_{i=1}^{N} \frac{\tau_i^{sc} + \tau_i^{ca}}{h_i}. \tag{5}$$

The network BU definition with network capacity consideration is

$$b_i = \frac{B_i \times 8}{BW \times h_i}, \tag{6}$$

where B_i is size of the data packet from Client i in bytes, BW is the capacity of bottleneck link in the network backbone in megabits per second (Mbps), and h_i is the time interval of data-packet transmissions in the network.

The following experiments are conducted to discuss differences among these definitions in Eqs. (3), (4), and (6). Detailed experimental setup and the DC motor specification can be found in Section 3.2. The network propagation delay $\tau_i^{sc} + \tau_i^{ca}$, tested by Ping, is about 0.48 ms. The processing time of each control iteration is about 0.866 ms. In this experiment, the size of data packet is 68 bytes. We assume that the capacity of the bottleneck link is 40 Mbps. Comparisons of various BU definitions are shown in Fig. 11. From Fig. 11, the NCS BU is the most conservative resource bound of an NCS. If the NCS BU can be guaranteed, the other two specifications will be guaranteed as well. Hence, the NCS BU can be applied as the primary resource for the allocation purpose. Hereafter, the NCS BU will be applied for the purpose of designing the bandwidth allocation algorithm and will be rewritten as BU for short.

Scheduling Algorithms

Roughly speaking, the scheduling problems in an NCS are to assign a transmission schedule to the sensors, actuators, and controllers on the network based on certain scheduling algorithms. A scheduling algorithm is a set of rules that determines transmission order of the data packets based on their execution times and deadlines. In general, a scheduling algorithm can be categorized as a static scheduling algorithm or a dynamic scheduling algorithm. A static scheduling algorithm assigns the tasks a fixed scheduling priority in the beginning of task execution. The scheduling sequence will not change while the system executes the given tasks. A dynamic scheduling algorithm redistributes or reorders the tasks' scheduling priorities or sequences during their

executions. The redistribution or reordering is usually based on certain given policies of the scheduler.

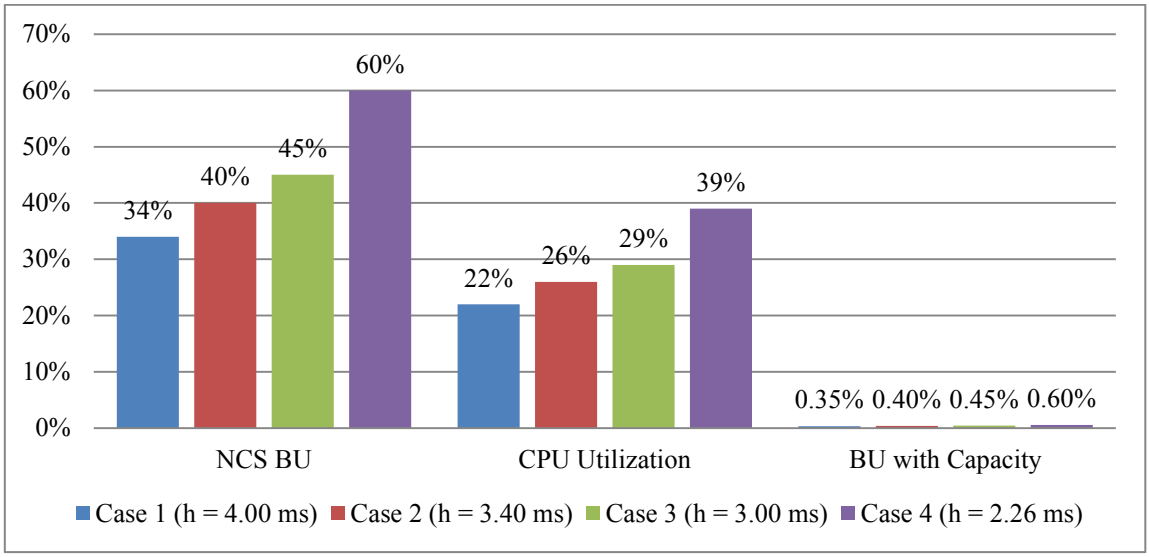


Fig. 11. Comparisons of various BU definitions

Table 2 gives comparisons of several existing scheduling algorithms. However, these scheduling algorithms could not promise a satisfactory performance of an NCS. Since bandwidth allocation and scheduling are correlated aspects in the NCS, a scheduling algorithm, which is dynamic and flexible to the BU, would be expected.

3.1.3. Clock-Driven and Event-Driven Tasks

Concepts of clock-driven and event-driven tasks are originally from the RTOS task scheduling. Here in the NCS, definitions of the clock-driven and event-driven tasks are borrowed and revised from the RTOS task scheduling.

Table 2. Scheduling algorithms comparisons

Algorithm	Description	RTAI[1] Compatible	Global Information	Schedulability[2]	Priority
First-In-First-Out (FIFO)	The serving processes are based on the order of the queue.	Yes	No	–	Dynamic priority
Round Robin (RR)	Executes tasks in turns within a preset time-slice defined by the scheduler.	Yes	No	–	Same priority
Earliest Deadline First (EDF)	Executes the task with the shortest deadline first.	No[3]	Yes	1	Dynamic priority
Rate Monotonic (RM)	All the fixed priories are preset prior to the run time.	No[4]	Yes	$n(2^{1/n}-1)$	Static priority

[1] RTAI stands for real-time application interface, and can be found in Section 3.2.
[2] The schedulability of EDF and RM can refer to [62].
[3,4] Could be supported with necessary revisions, hard and easy to implement, respectively.

A clock-driven task's update or execution decision is made at a specific time instant. The time instant is decided prior to the system's implementation. In the NCS, the plant outputs of a client are generally considered as clock-driven tasks, which are strictly executed at each sampling period. The plant outputs of an NCS are not affected by the time delays and packet losses in the network. The plant outputs will be generated at each sampling period in an almost guaranteed manner.

An event-driven task's update or execution decision is made after certain specified events take place. The control-input calculations of Server and the actuator updates of Client in the NCS can be considered as event-driven tasks. The control-input

calculation of Server will be activated after the sensor-feedback packets of Client arrives at Server. If the sensor-feedback packets are lost in communication, Server will be unable to calculate the control inputs during the current control iteration because of lack of the current information of the plant. Similarly, the actuator update of Client can only be executed after the control-input packets from Server arrive at Client.

Table 3 gives a clear view of each category of the tasks in the NCS. The hosting node of each data packet and its execution conditions are also given.

Table 3. Type definition of the tasks in an NCS

Name of task	Type of task	Hosting node	Execution condition
Plant outputs	Clock-driven	Client	Sampling period
Control-input calculation	Event-driven	Server	Arrival of plant-output packet
Actuator updates	Event-driven	Client	Arrival of control-input packet

3.2 EXPERIMENTAL SETUPS

3.2.1. Hardware Setups

As shown in Fig. 9, Client 1, the ball maglev system levitates a steel ball at a 4-mm equilibrium position, which is measured from the bottom of electromagnet to the top of steel ball. Client 2, 4, 5, and 6, the DC motor speed-control systems, maintain the speed of the DC motors at 10 resolutions per second (rps). Client 3, the wireless autonomous robotic wheelchair, follows a pre-set path or explores an unknown

environment with its real-time path-planning capability. These three clients require various levels of sampling frequencies to maintain their stability and system performance. Client 1 requires a fast sampling frequency; Client 3, a slow sampling frequency; while Client 2 is in the middle. This structure brings more challenges to the NCS due to various requirements from the combination of fast, medium, and slow dynamic systems.

Ball Maglev System

Client 1 is the ball maglev system shown in Fig. 12 [63]. In order to levitate the steel ball at a predetermined steady-state equilibrium position with an electromagnet, the ball maglev system consists of a personal computer (PC), a position sensor, a pulse-width modulation (PWM) power amplifier, and power supplies to drive a light bulb and an electromagnetic actuator. The optical position sensing unit consists of an incandescent light source, a CdS photocell, and a 15-V DC power supply.

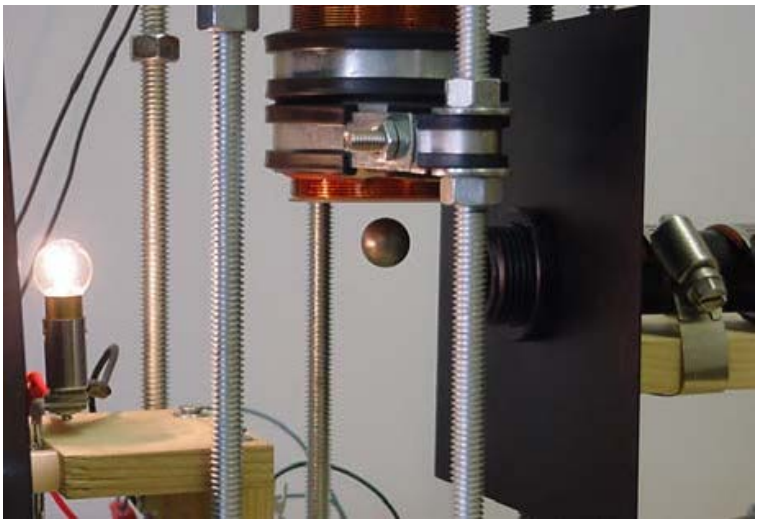


Fig. 12. Ball maglev system

DC Motor Speed-Control Systems

The DC motor speed-control systems are shown in Fig. 13 [64]. The speed of the DC motor is directly proportional to supplied voltage, which is fed to a PWM amplifier. This drives the motor at a speed depending on the commanded voltage. The shaft angular displacement per unit time is sampled using an encoder. A PCI-6221 data-acquisition (DAQ) card by National Instruments (NI) enables the test bed to send out sensor-feedback data packets and receive control-input data packets through the LAN.

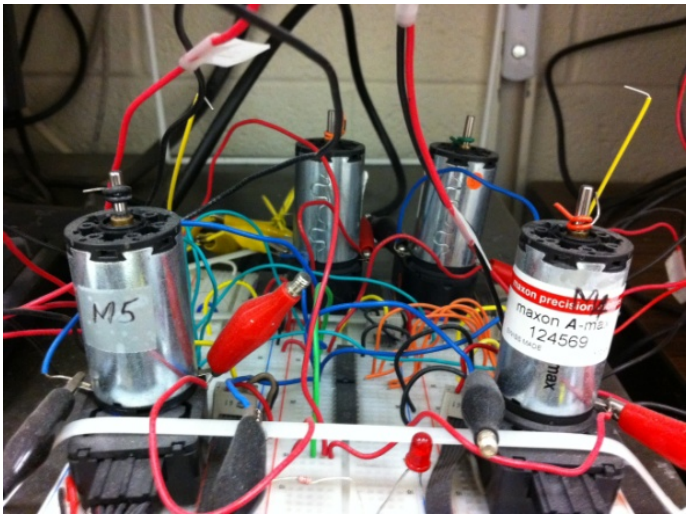


Fig. 13. DC motor speed-control systems

Wireless Autonomous Robotic Wheelchair

The wireless autonomous robotic wheelchair in Fig. 14 is constructed on the frame of an Invacare Ranger II^{TM} electric powered wheelchair with the length, width, and height of 70 cm, 48 cm, and 55 cm, respectively [65]. It is capable of supporting a weight of approximately 100 kg. Two independent 12-V DC motors are the actuators to

drive front wheels. The front wheels' diameter is 31.75 cm. The speed of the motors is controlled by the output voltage of the PWM amplifiers on board of two Diverse Electronic's modular MC-7 motor controllers. In the rear are two 18-cm-diameter caster wheels. A NI USB-6501 DAQ card performs all data-acquisition and control functions. Three Sharp GP2D15 and two Sharp GP2D12 infrared distance-measuring sensors, mounted on a sensor bracket in front of the wheelchair, are used to detect obstacles in the path. The GP2D15 detects obstacles at a fixed range of 24 cm, and the GP2D12 detects obstacles at a range from 12 cm to 80 cm. Seven PDV-P5001 CdS photocells manufactured by Advanced Photonix are assembled to equip the robotic wheelchair with capability of tracking a specific light. All the seven photocells are distributed evenly by 22.5° on the bracket.

Fig. 14. Wireless autonomous robotic wheelchair

The control system of the wireless autonomous robotic wheelchair is shown in Fig. 15. Client collects the sensor signals from the photocell, the infrared sensors, and

the Hall-effect sensors through the USB-6501 card. Then, the client laptop encapsulates the sensor data into one single data packet and sends it to Interoperability Suite through the WLAN. Interoperability Suite repacks the sensor data packet with necessary header changes, and then transfers the sensor-feedback data packet to Server. The control-input data packet will be sent back to Client through Interoperability Suite following the similar pattern.

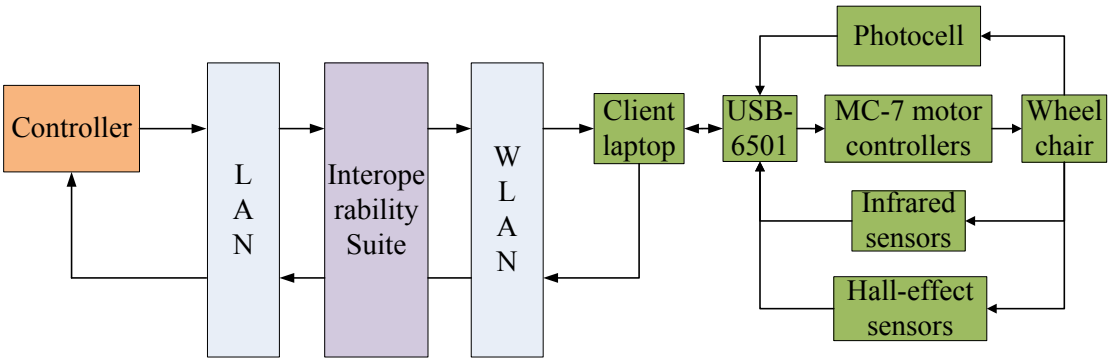


Fig. 15. Robotic-wheelchair control system

3.2.2. Software Setups

To ensure the real-time operation of Server, Linux with RTAI [66] is found as a competitive OS environment. RTAI modifies the Linux kernel to make it a real-time operating environment. RTAI offers basically the same functionalities as a Linux kernel core, but adding features of a real-time OS. Linux Redhat 7.3 with RTAI 3.4 is chosen to be the OS running on Server, and Linux Ubuntu 6.10 with RTAI 3.4, on Client 1, 2, 4, 5 and 6. The control and measurement device interface (Comedi) [67] is applied as drivers

and libraries of data acquisition on Clients. The programs of Server and Clients 1 and 2 are developed in C. The programs of Client 3 are operated on a Dell Insprion 1525 laptop computer with an Intel Core 2 Duo T7250 2.00 GHz processor and 4 GB RAM. The programs of Client 3 are developed by Visual Basic and Visual C++ 2008 on the Microsoft Windows XP OS. The programs of Client 3 are built on Windows XP while Server program is built on Linux. Since Windows and Linux cannot communicate directly, Samba [68] is chosen to be Interoperability Suite in Fig. 9. Samba 5.0 is installed on a Dell GX240 desktop computer running with Ubuntu 6.10 with RTAI 3.4 as Interoperability Suite. The programs for Server, Clients, and Interoperability Suite are listed in Appendix A.

3.2.3. Network Protocols and Data-Packet Structures

UDP provides a datagram service that emphasizes reduced latency over reliability. It is a connectionless protocol. A datagram can be sent at any moment without preparations. UDP does not guarantee that the datagram will be delivered to its destination host. The datagram can also be delivered in an incorrect order. Although UDP is unreliable, it has fewer headers than Transmission Control Protocol (TCP). UDP does not have retransmission mechanism as TCP. However, the UDP connection is faster and introduces smaller propagation delays to an NCS. For instance, if one packet in the network cannot reach its destination node at its first try, no retransmission of the data packet will be done with UDP protocol because of lack of the acknowledgment message. Then to the NCS, this specific data packet is lost. The UDP protocol does not try to recover the losses of data packets, so that transmissions of the next available data

packets will not be affected. However, for TCP with the acknowledgment and retransmission mechanism, this data packet will be retransmitted for a certain amount of times (3 times for the current TCP/IP protocol suite) until the retransmission threshold is reached. All the retransmissions take time and will put the other data packets in a buffer for a certain amount of time. Therefore, the time delays of other data packets will be longer than expected. The trade-off between the TCP and the UDP for an NCS is reliable connections and longer propagation delays vs. less reliable connections and faster transmissions. Depending on specifications and system requirements of various NCS applications, UDP can be a reasonable choice as a suitable protocol. For some NCSs, UDP is a preferred protocol for better performances [67]. Due to real-time characteristics of our NCS, UDP is chosen to be the protocol for experiments. More details regarding to comparisons of TCP and UDP and reasons to choose the UDP as the communication protocol can be found in [28, 69–70].

Data-packet structures of Server, Client, W Client (wireless client) and Interoperability Suite are given in Fig. 16. The 802.3 header, 802.1x header, IP header, and UDP header are standard Internet protocol headers. Control data and sensor data segments are data segments generated by Server and Client, respectively. Timestamp is set up by Client to track total time delays and execution times in the current control iteration. Identifier segment is to identify Client for data packets matching purpose. BU segment contains current BU information of the clients. Type segment is used to identify whether a client has a fixed sampling frequency or a variant one. SP segment contains a new sampling period assigned to each client if applicable.

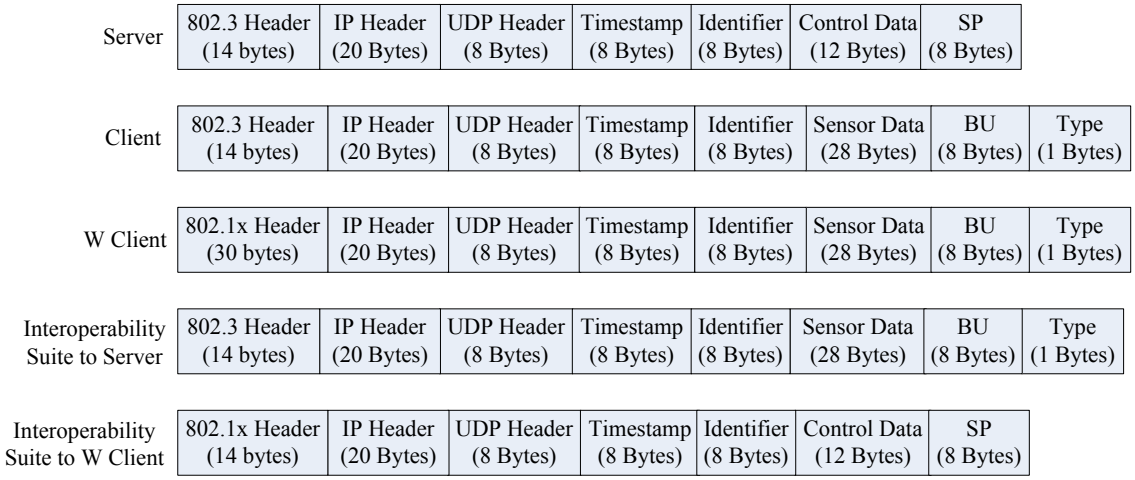


Fig. 16. NCS data-packet structures

3.2.4. NCS Control Flows

A flow chart of the control algorithm on Server is given in Fig. 17. These Clients send the sensor measurements to and request the control inputs from Server. Server responds all the requests by their coming-in sequences and their identification numbers. If the total BU (TBU) of the NCS is less than an upper bound of current available bandwidth, Server calculates the control inputs and sends them back to each client directly or via Interoperability Suite. The upper bound may vary depends on the scheduling algorithm running on the system and the available bandwidth in the NCS. If the TBU is greater than the upper bound, Server checks Client's sampling period type and calculates the control inputs directly for Clients that have fixed sampling periods. If Clients have variant sampling periods, Server will increase their current sampling periods by 5 % and check the TBU again until the TBU is no longer greater than the

upper bound. To maintain the stability of each client, a maximum sampling period $h_{\max}$ will be set as a boundary of the dynamic sampling period algorithm in the NCS. After the control inputs have been calculated, Server sends the control inputs to each client directly or via Interoperability Suite. Then Clients can update the actuators with the latest control inputs if data packets losses do not take place in data transmission. Note that not all the requests from Clients can be executed in time because of the stochastic nature of the NCS. Some data packets may be lost in data transmission so that the updated control inputs may not available to Clients all the time. In Section 4, one can see that predicted data will be applied to the NCS if data packets are lost in the transmission.

3.3 ANALYTICAL RESULTS

3.3.1. Off-Line Clock Synchronization

Since all the nodes in an NCS operate on different clocks, accurate time delays of data transmission among the nodes cannot be measured. Even when all the nodes' clocks are initially set accurately, real clocks will differ after a while due to clock drifts caused by clocks counting time at slightly different rates. However, an accurate measurement of the time delays may be unnecessary if the controller is robust enough to the time delays. Based on the earlier assumption of the clock resolution, all the nodes in this research are assumed to have the same clock resolution and no clock drifts will exist. One simple off-line clock synchronization method will be presented to provide a relationship of clocks on different nodes.

To understand this synchronization method, take one sampling period for example as shown in Fig. 18.

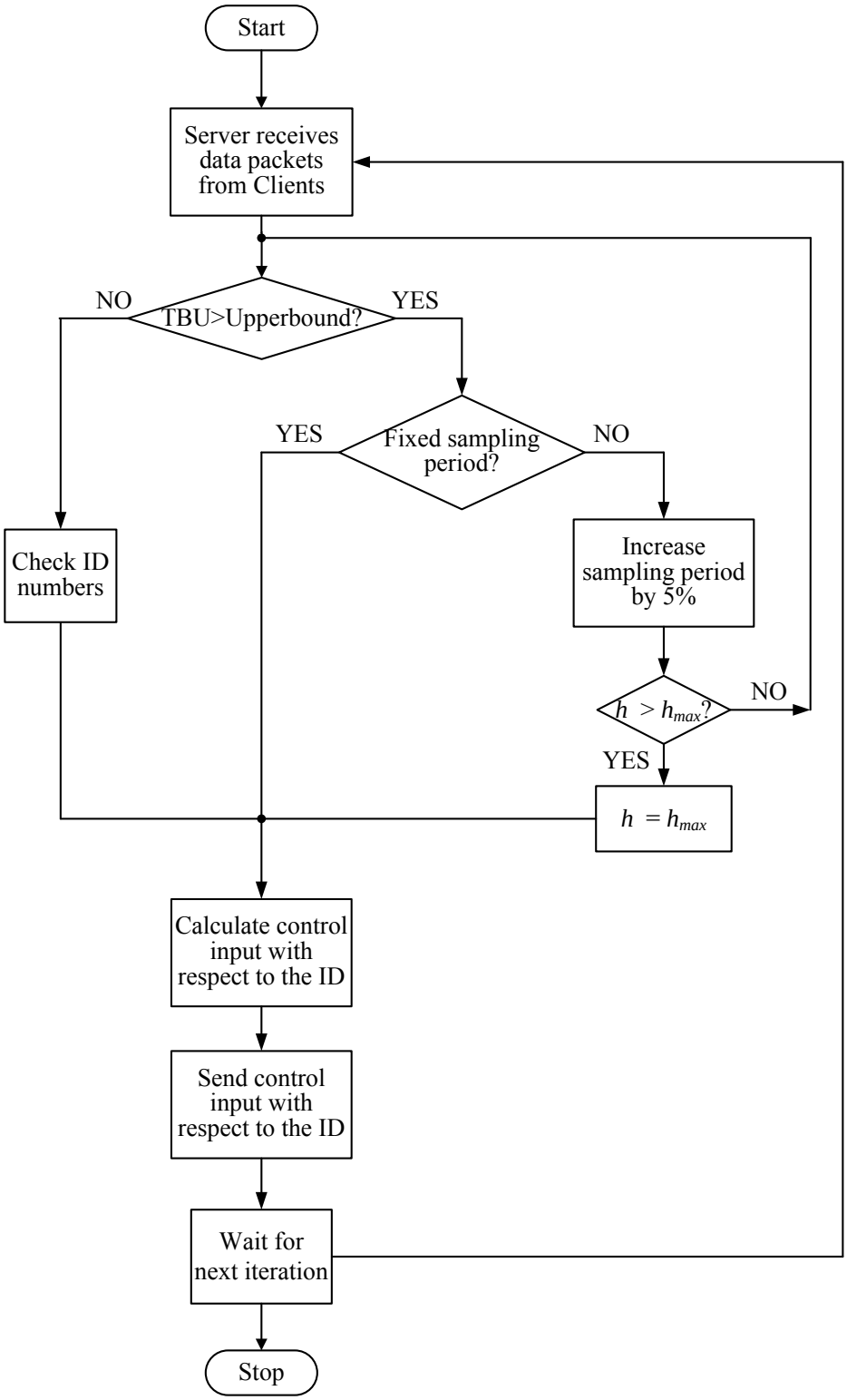


Fig. 17. Flow chart of the multiscale NCS control architecture

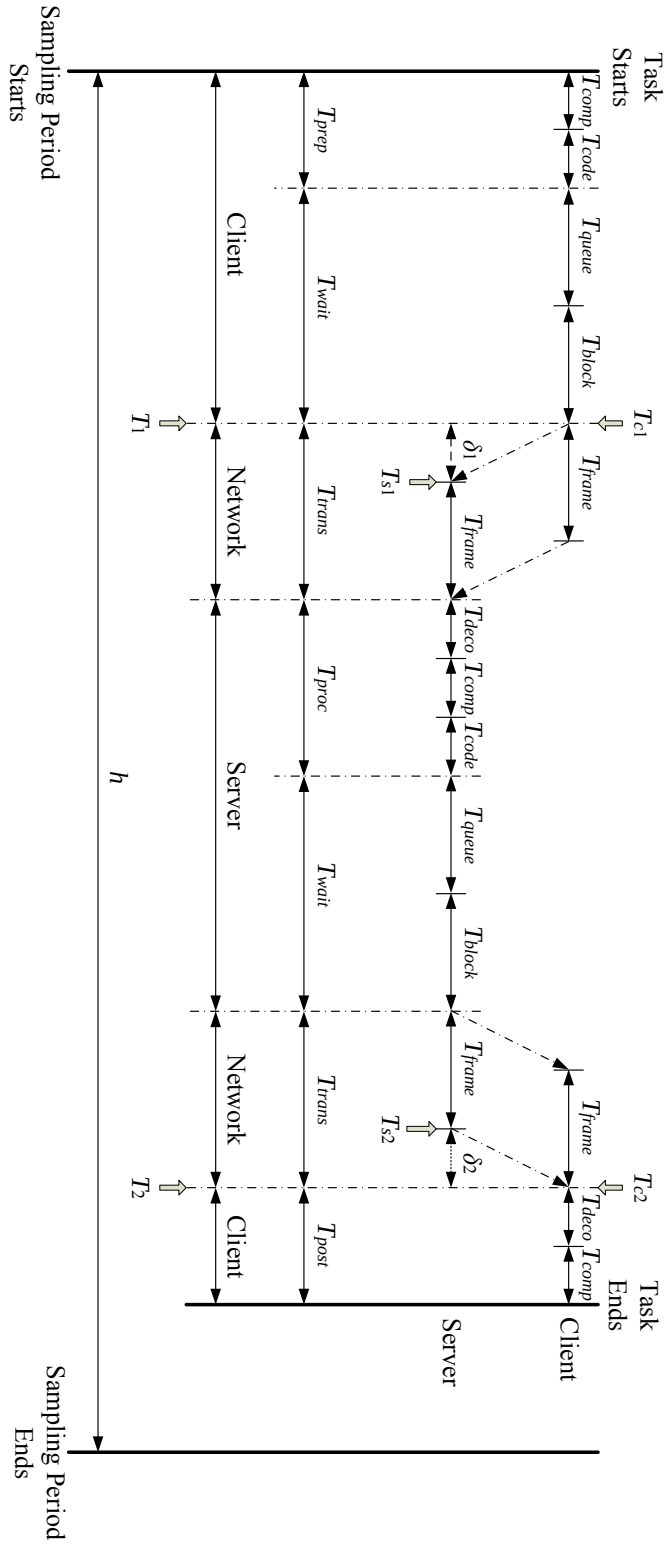


Fig. 18. Detailed timing diagram of one sampling period in the NCS

As shown in Fig. 18, in the beginning of this sampling period, the control task starts with collecting and coding sensor measurements (encapsulate a data segment) on Client. Then the data packet with sensor measurements and necessary headers will be held in a queue waiting for transmission. If the network is idle, the transmission request will be permitted. The data packet will be decoded after it arrives at Server. The sensor-feedback information carried by the data packet will be applied for the control-input calculation. Later, the control inputs will be coded into data segments, queued in a buffer, and transmitted back to Client. Client receives the control inputs and finishes up the current control iteration with decoding and executing the data packet. The control task should be accomplished within the current sampling period to guarantee the stability and system performance of the NCS. However, the control task could be delayed for more than one sampling period because of the network condition or fail because of unexpected longer time delays, packet losses, or other uncertainties.

To start with the off-line clock-synchronization method, a timestamp is inserted at the marked time T_1 on Client as in Fig. 18. When the control-input data packet arrives at Client, the timestamp carries T_1 will be checked at time T_2. One can calibrate these timestamps as T_{c1}, T_{c2}, T_{s1}, and T_{s2} on Client and Server, respectively. T_{c1} is the time instant that the first byte of the data packet leaves Client, and T_{c2}, the time instant that the last byte of the data packet arrives at Client. T_{s1} is the time instant that the first byte of the data packet arrives at Server, and T_{s2}, the time instant that the last byte of the data packet leaves Server.

Although Client and Server run on separate clocks, the absolute time instant in real world should be the same so that $T_1 = T_{c1} = T_{s1} - \delta_1$ and $T_2 = T_{c2} = T_{s2} + \delta_2$. For simplicity, one can assume that $\delta_1 = \delta_2 = \frac{\Delta T_c - \Delta T_s}{2} = \frac{(T_{c2} - T_{c1}) - (T_{s2} - T_{s1})}{2}$. ΔT_c and ΔT_s can be obtained on each node with certain timestamp calculations. The two timestamp coordinates $(T_{c1}, \hat{T}_{s1})$ and $(T_{c2}, \hat{T}_{s2})$ will be available for each control iteration with $\hat{T}_{s1} = T_{s1} - \delta_1$ and $\hat{T}_{s2} = T_{s2} + \delta_2$. These timestamp coordinates can be applied to calculate the relation between Server and Client with a least-square method (LSM).

From above discussion, the off-line clock synchronization is obtained from analytical experiments as follows. With four different sampling periods and the LSM, the linear relation between Server and Clients in this research is given by

$$T_s = 1.001 T_c - 1.6027 \times 10^6 . \tag{7}$$

This polynomial approximation is linearized by the LSM at a 3.4-ms sampling period and is verified at various sampling periods. The constant in Eq. (7) may vary with the time interval between the instants the OS is booted up and the OS is terminated. It indicates the linear relation will have to be recalibrated after the OS reboots. Note that the unit of the polynomial approximation in Eq. (7) is ns. Therefore, the proportional coefficient of 1.001 in Eq. (7) indicates an approximate 1 μs clock difference between Server and Client, which verifies the assumption about the clock resolution.

Figures 19–22 illustrate the polynomial approximation in Eq. (7) and experimental results with the 2.267-ms, 3.4-ms, 6.8-ms, and 15.1-ms sampling periods,

respectively. The experimental results collect two timestamp coordinates $(T_{c1},\ \hat{T}_{s1})$ and $(T_{c2},\ \hat{T}_{s2})$ as indicated in the figures. From Figs. 19–22, one can see that Eq. (7) could catch practical relation of the clocks on Server and Client.

Another way to estimate the time delays in an NCS is to set up the timestamp segment in the data packets, the total time delays and packet losses can be detected by Client at the end of each sampling period. The total time delays can be inferred by calculating the difference between the time instance Client sends sensor-feedback packets to Server and the instance Client receives control-input packets from Server. This structure of the total time delays has the following form

$$\tau = \Delta timestamp = \tau^{sc} + \tau^{ca} + \tau^{p}. \tag{8}$$

Note that, in general, τ^{ca} and τ^{sc} are not necessarily the same. Without a clock-synchronization mechanism, exact τ^{ca} and τ^{sc} are unavailable, and we can simply assume that

$$\tau^{sc} \approx \tau^{ca} \approx \frac{1}{2}\tau. \tag{9}$$

This is possible because one can expect that the propagation delay in the sensor-to-controller and controller-to-actuator links should be the same if no packet losses or major uncertainties take place in the network. Compared to the time delays over the network, the processing time can be smaller or neglected under certain circumstances.

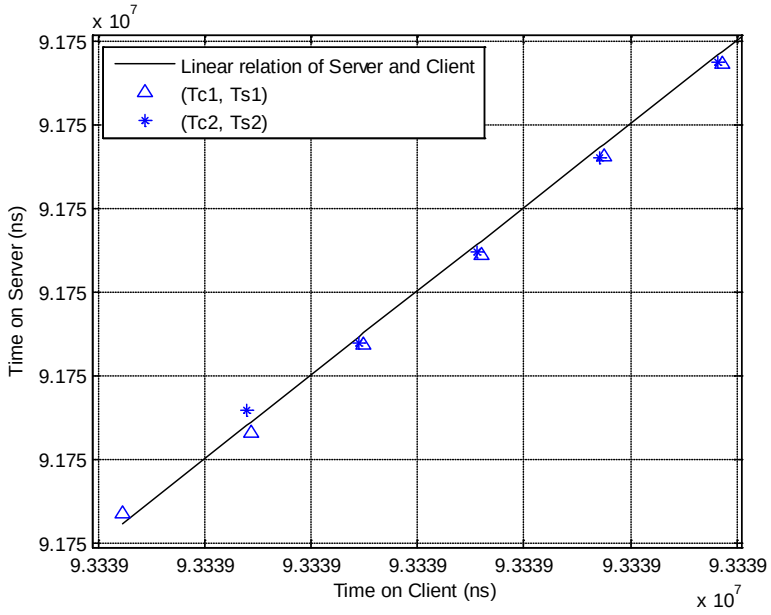


Fig. 19. Polynomial approximation and experimental data with the 2.267-ms sampling period

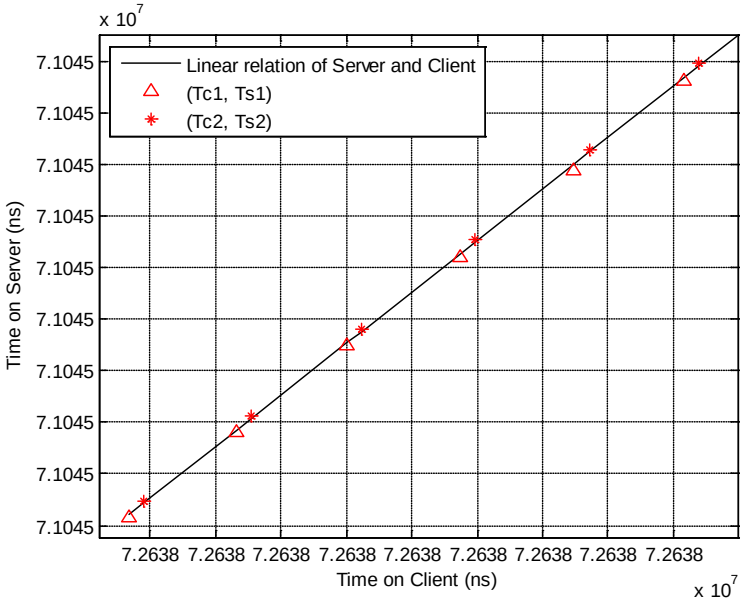


Fig. 20. Polynomial approximation and experimental data with the 3.4-ms sampling period

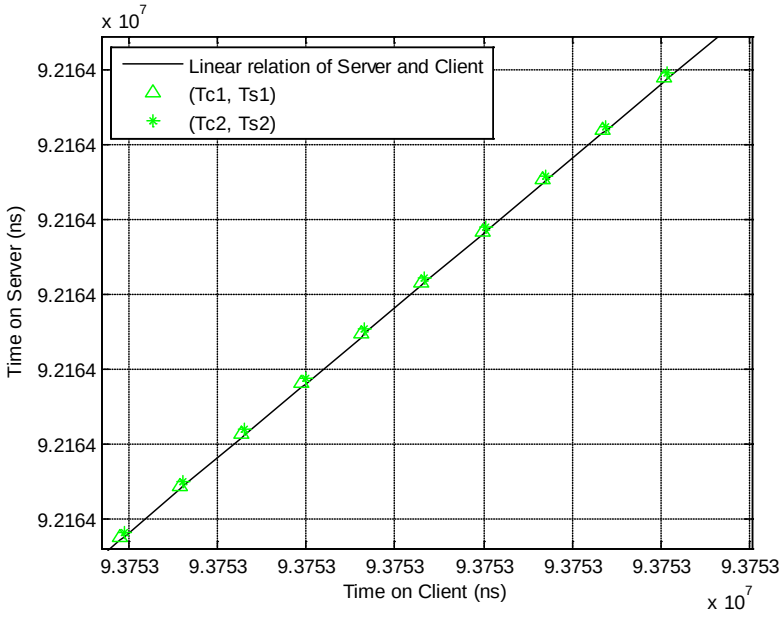


Fig. 21. Polynomial approximation and experimental data with the 6.8-ms sampling period

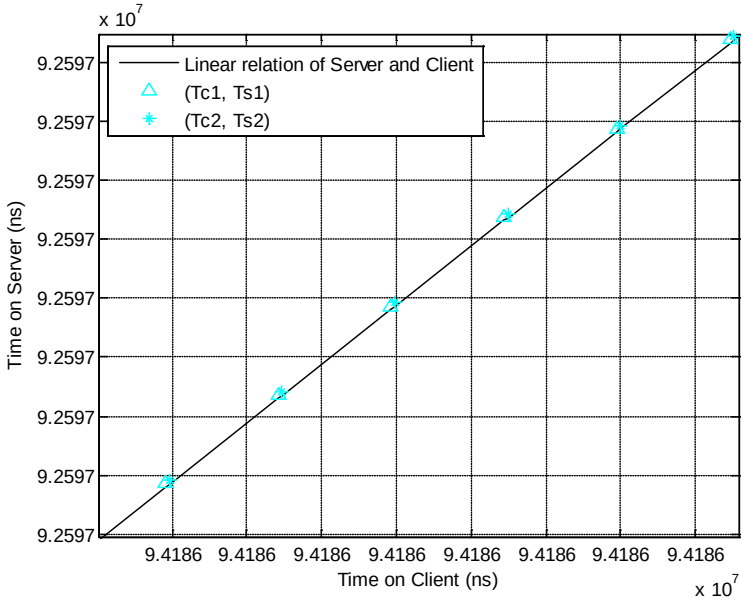
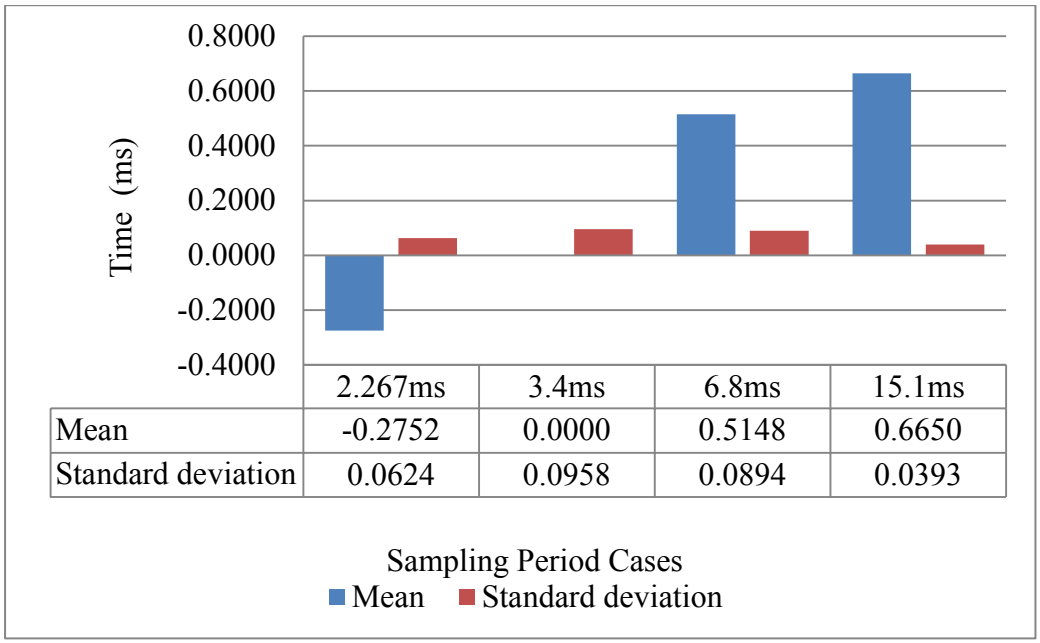


Fig. 22. Polynomial approximation and experimental data with the 15.1-ms sampling period

Figure 23 shows mean and standard deviation of differences between the time delays calculated by Eqs. (7) and (8) with four different sampling periods. Equation (7) can yield explicit time delays in the sensor-to-controller and controller-to-actuator links compared to Eq. (8) within the current control iteration. Equation (8) cannot generate explicit time delays τ^{ca} and τ^{sc}, and can only calculate τ for Server in the next control iteration, but it is easier to be implemented in algorithm compared to Eq. (7). From the analysis, the difference of these two time-delay calculations are quite small so that either one can be applied to the experiments of an NCS. However, for the simplicity of the algorithm implementation, Eq. (8) will be chosen to calculate the time delays in the NCS. This may also save calibration and calculation times of Eq. (7) so that the total time delays may be reduced.



	2.267ms	3.4ms	6.8ms	15.1ms
Mean	-0.2752	0.0000	0.5148	0.6650
Standard deviation	0.0624	0.0958	0.0894	0.0393

Fig. 23. Statistics of the time delays with various sampling periods

3.3.2. Time-Delay and Packet-Loss Experiments

As discussed earlier, time delays and packet losses are two of unique properties of the NCS brought by the network. Therefore, the time delays and packet losses should be considered as crucial parts of design of an NCS. Because of the nature of the network, the time delays and packet losses can be stochastic in an NCS. Various levels of time delays and packet losses can exist in the NCS. Each different level of time delays and packet losses will also have different effects on the stability and system performance of an NCS.

Client 2 is adopted here to test the effects of the time delays and packet losses on the system performance of an NCS. The transfer function of Client 2 is

$$G(s) = \frac{20.2}{9.92s + 2.57}. \tag{10}$$

A proportional-integral (PI) controller is applied to control Client 2 as

$$D(s) = \frac{1.5s + 5}{s}. \tag{11}$$

The time-delay experiment is performed for 20,000 iterations with Client 2 with a 3-ms sampling period. Figure 24 shows the first 2,000 iterations. From Fig. 24, the mean and standard deviation of the time delays in the network are about 0.5034 ms and 0.0414 ms, respectively.

The DIAE is adopted to be the performance index of the NCS and formulated as follows [71]

$$DIAE = \sum_{k_0}^{k_f} |e_k|, \tag{12}$$

where k_0 and k_f are the initial and final times of the interval of interests, and e_k is the error between the actual and reference signals.

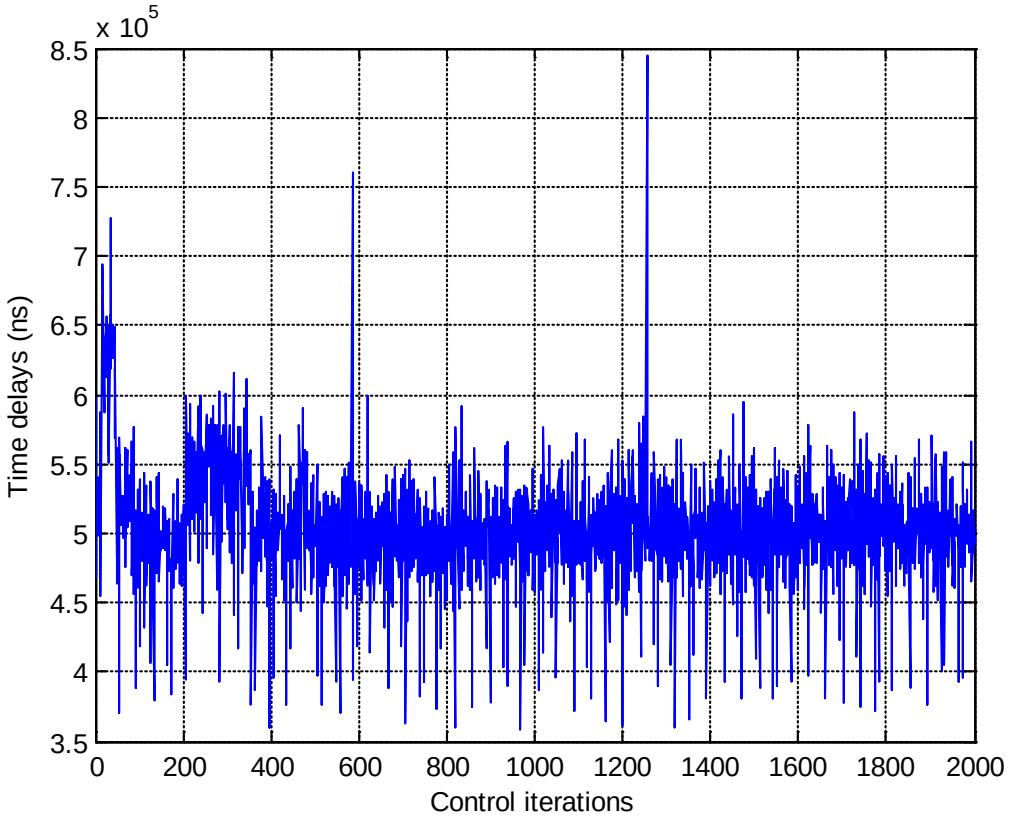


Fig. 24. Time delays in the NCS

Since the actual time delays in the network are not controllable variables in the NCS design, we manually add more time delays in the program to discuss the

performance of an NCS with various time delays. To show the effects of the different levels of time delays on the NCS, the following six sets of experiments are conducted, the experiments with extra 0-ms, 0.2-ms, 0.4-ms, 0.6-ms, 0.8-ms and 1-ms delays. Note that the time delays in the NCS are the sum of τ and the extra given delays. The system performances of Client 2 with 5-ms, 10-ms, and 15-ms are given in Figs. 25–27. From these figures, the DIAEs of Client 2 increase as the sampling periods and time delays increase in the NCS. As the time delays increase by 0.2 ms, the DIAE of Client 2 increases about 3–5%. Note that occasionally the DIAE of Client 2 with a given sampling period with longer time delays may be smaller than the one with shorter time delays as shown in the figures. This can be caused by uncertainties in the network. Figure 28 shows the DIAE of Client 2 with respect to the sampling periods and the time delays in 3-D. Note that the relation among the DIAE, sampling periods and time delays is no longer linear because of the complex dynamics of the network. As shown in Fig. 28, the DIAE decreases and then increases as the sampling period increases. Also, the DIAE increases as the time delays increase. This verifies that longer time delays and shorter sampling periods inevitably degrade the system performance.

Figures 29–31 show the DIAE of Client 2 with 5-ms, 10-ms, and 15-ms sampling periods with various packet losses. The DIAEs of Client 2 increase as the sampling period and packet losses increase in the NCS. As the packet losses increase by 10%, the DIAE of Client 2 increases about 10–15% if the packet losses are less than 50%. Note that there might exist discrepancies caused by the uncertainties in the network.

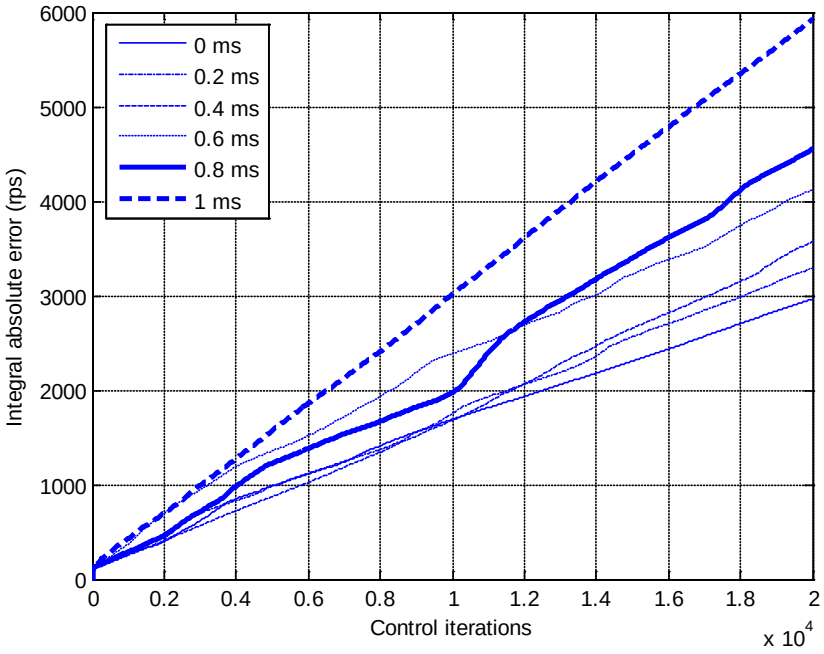


Fig. 25. DIAEs of Client 2 with a 5-ms sampling period with various time delays

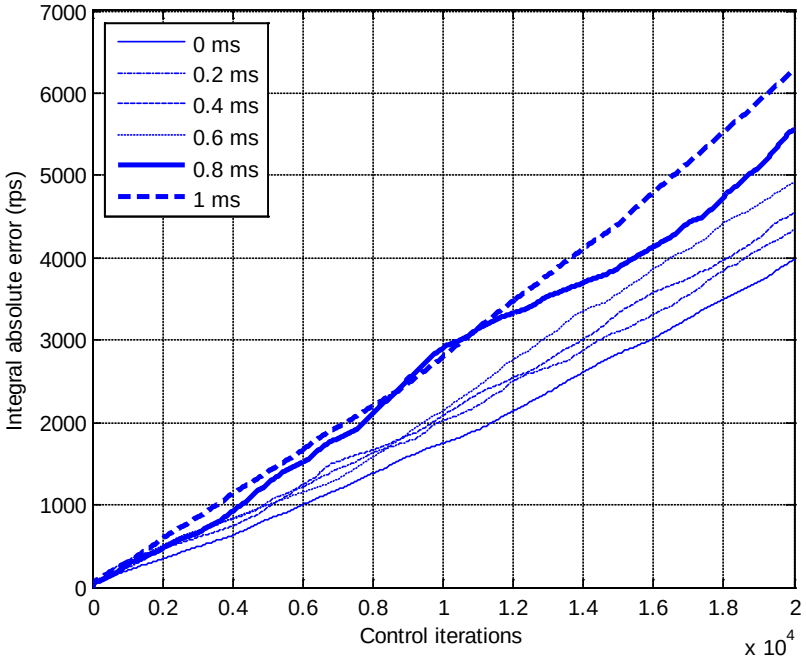


Fig. 26. DIAEs of Client 2 with a 10-ms sampling period with various time delays

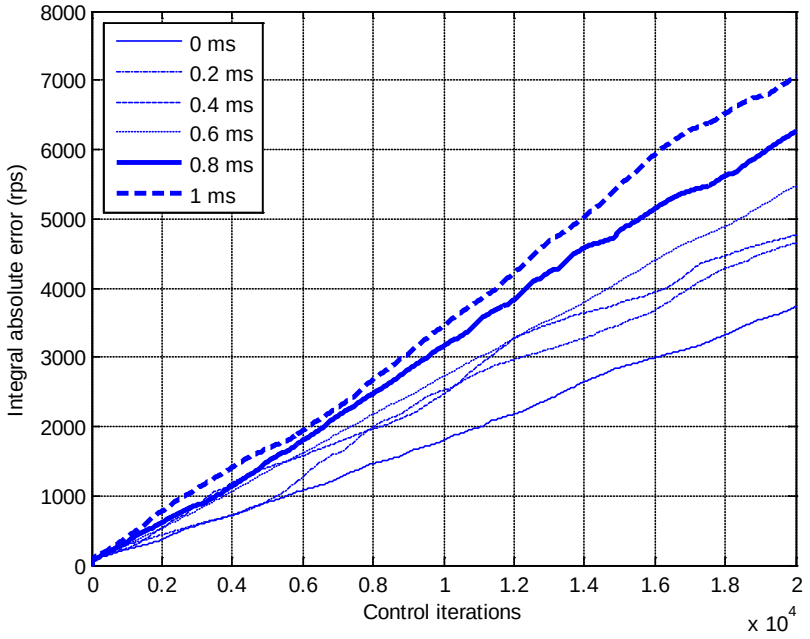


Fig. 27. DIAEs of Client 2 with a 15-ms sampling period with various time delays

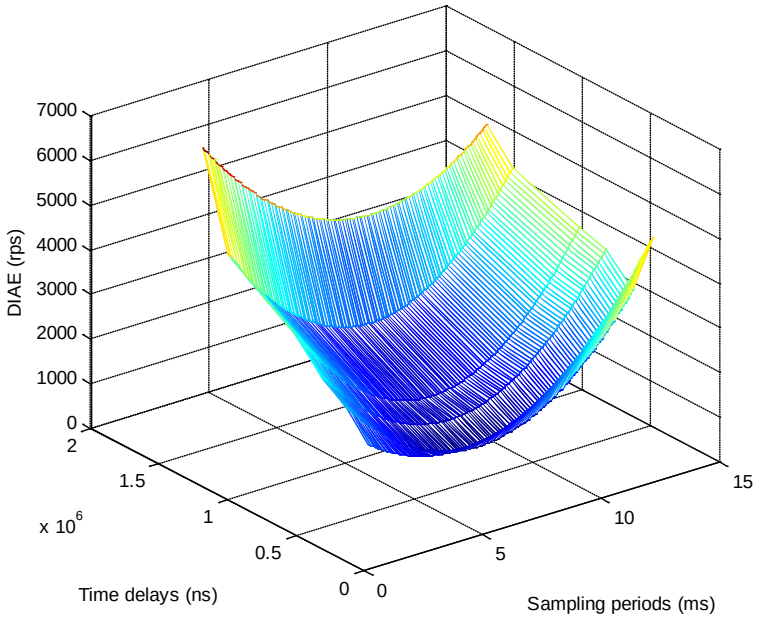


Fig. 28. DIAE vs. the sampling periods and the time delays

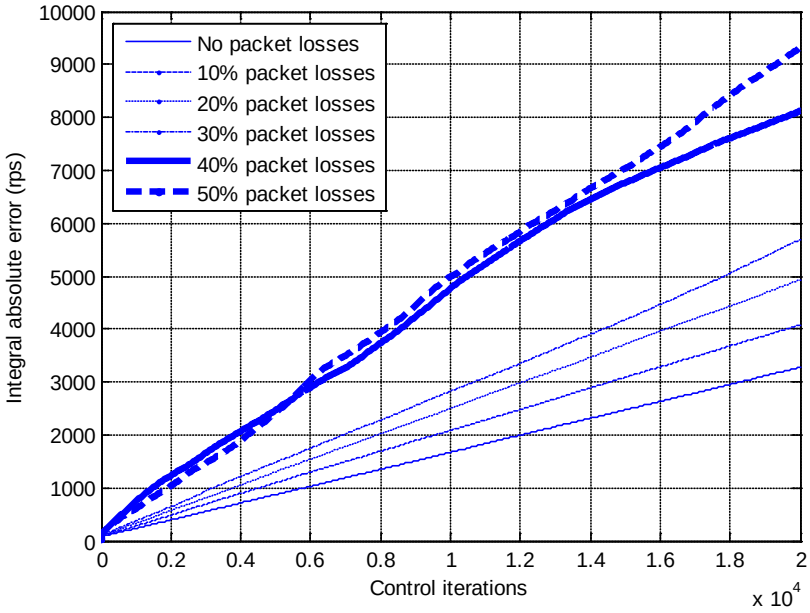


Fig. 29. DIAEs of Client 2 with a 5-ms sampling period with various packet losses

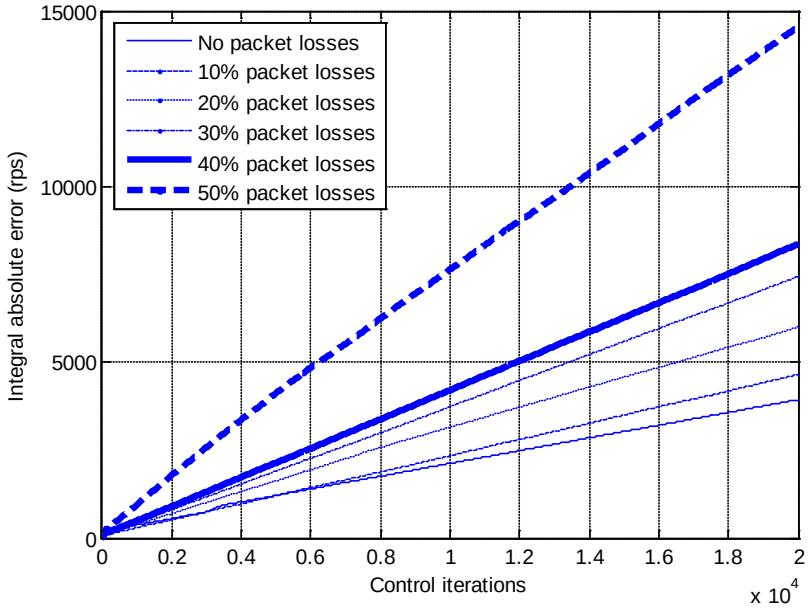


Fig. 30. DIAEs of Client 2 with a 10-ms sampling period with various packet losses

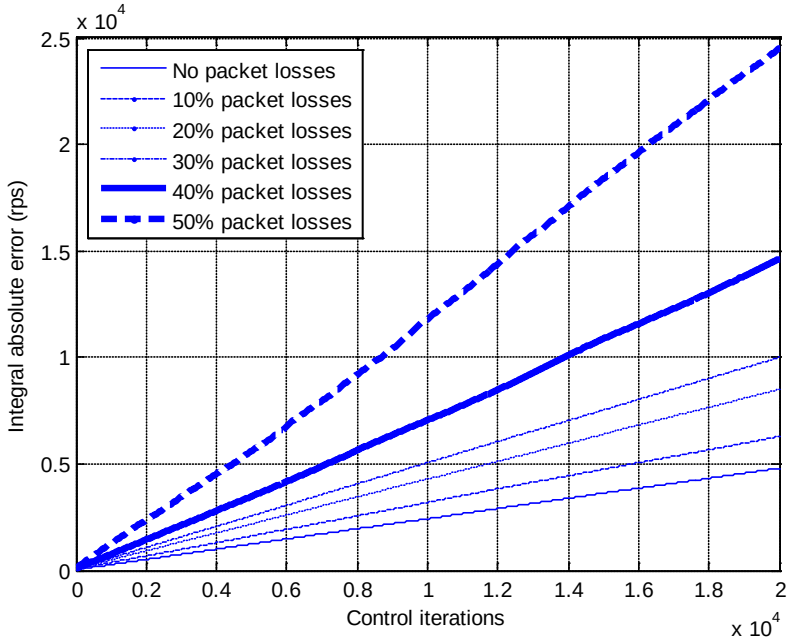


Fig. 31. DIAEs of Client 2 with a 15-ms sampling period with various packet losses

3.3.3. Bandwidth-Allocation Experiments

As shown in Fig. 9, three different types of clients are involved in the NCS experiments. The number of clients in the NCS is not strictly limited to three. However, complexity of the NCS will increase by adding more clients. The maximum number of clients can be determined by available network resources and the minimum BU requirement of each client. For instance, if each client requires 30% network bandwidth, the NCS could incorporate three clients with 10% idle network bandwidth.

The performance of an NCS mainly depends on the time-delay and packet-loss level and resource allocation and scheduling. The number of clients may not have direct effects on the NCS, but it is indeed an important parameter of an NCS in the sense that

more clients would need to compete for the network resources and decrease the schedulability. In this case, more robust controller may be necessary to the NCS, but the basic structure of the NCS will not change. Note that Client 1 is an open-loop unstable system that requires a fast fixed sampling period to maintain its stability. Client 2 is open-loop stable system that can have variant sampling periods. Client 3 is connected to Server via WLAN. Therefore, the NCS in this research includes Clients that are either wired or wireless systems, and open-loop unstable or stable systems with fast, medium, and slow dynamics. These Clients can have fixed or variant sampling periods. All these configurations bring more complexity to the NCS. Clients employed in this NCS can be replaced by other dynamic systems. In this research, we include these three typical dynamic systems to verify the feasibility and the performance of the multiscale NCS with wired and wireless frameworks.

Client 1, the ball maglev system, is an open-loop unstable system and requires a fast sampling period to guarantee the stability and the performance. For this reason, a 3-ms sampling period is assigned to the system with following controller [63]

$$\begin{aligned} u(k) &= 0.782u(k-1) + 0.13u(k-2) \\ &\quad -41500.0e(k) + 48779.1e(k-1) - 31913.5e(k-2), \end{aligned} \tag{13}$$

where $u(k)$ is the control input and $e(k)$ is the error.

The discrete-time controller of Client 2, the DC motor, is as follows [64]

$$u(k) = u(k-1) - (1.5 - 2.5h_2)e(k) + (1.5 + 2.5h_2)e(k-1), \tag{14}$$

where h_2 is the sampling period given in Table 4.

As discussed before, the BU threshold of the NCS depends on the implemented scheduling algorithm. Here, EDF is adopted for the experiments, which gives the NCS 100% BU threshold [60]. Table 4 presents four different BU combinations and their corresponding sampling periods (h_*) and BU (b_*) of each client in the NCS. In Table 4, Case 4 exceeds the BU threshold. All the other three cases are within the BU threshold.

Figures 32–35 illustrate the system performance of each client for the four cases of the NCS with wireless client, respectively. Figure 36 shows the system performance of each client of Case 4 without dynamic sampling period assignment algorithm in Fig. 17. Each figure shows the performances of the ball maglev system, the DC motor, and the autonomous robotic wheelchair as parts (a), (b), and (c), respectively. Each part contains the same number of the samples (10000, 2000, and 200 samples for the ball maglev, DC motor, and robotic wheelchair, respectively) according to the sampling periods given in Table 4.

Table 4. Four cases of experiments with the corresponding sampling periods and BUs

Case	Client 1		Client 2		Client 3		TBU
	h_1	b_1	h_2	b_2	h_3	b_3	
1	3 ms	43.5%	5 ms	27.0%	100 ms	18.95%	84.95%
2	3 ms	43.5%	10 ms	13.5%	150 ms	12.63%	69.63%
3	3 ms	43.5%	15 ms	9.0%	300 ms	6.3%	58.8%
4	3 ms	43.5%	3 ms	45.0%	80 ms	23.6%	112.1%

From Fig. 32, although the TBU of Case 1 did not exceed the threshold, the performance of each client was degraded compared to Cases 2 and 3 in the Figs. 33 and 34, respectively. It is because the sampling periods of Clients 2 and 3 in Case 1 were smaller than the one in Cases 2 and 3. Therefore, more data packets were exchanged in the network. It would introduce longer time delays or even packet losses to the NCS so that the system performance could be degraded. From Fig. 35, the TBU was greater than 100% when Client 3 joined the experiment around 2 s. Based on the algorithm in Fig. 17, the sampling periods of Clients 2 and 3 were increased by 5% each time until the TBU was on longer greater than 100%. Note that in Fig. 35 there was a performance degradation of Client 2 around 2 s when Client 3 joined the NCS. The sampling periods of Clients 2 and 3 were eventually reset as 3.83 ms and 102.10 ms, respectively. The TBU was reduced to 97.53% for Case 4. To show the effectiveness of the control flow in Fig. 17, the experiment of Case 4 was conducted again without the control flow algorithm. The trajectories of each client were shown in Fig. 36. From Fig. 36, the BU threshold was exceeded, and Clients 1 and 3 failed. For Client 1, the steel ball could not be levitated. The 10-mm equilibrium-like position in Fig. 36 was the sensing limit of the photocell of the ball maglev system, not the actual position of the steel ball. Although Client 1 could not be levitated, it still sent the sensor measurement packets to Server every 3 ms. Therefore, Client 1 failed did not release any computational and network resource to other clients. For Client 3, the wheelchair could not track its pre-set straight path. Although Client 2 was still stable, it could not track the reference signals faithfully as the other three cases.

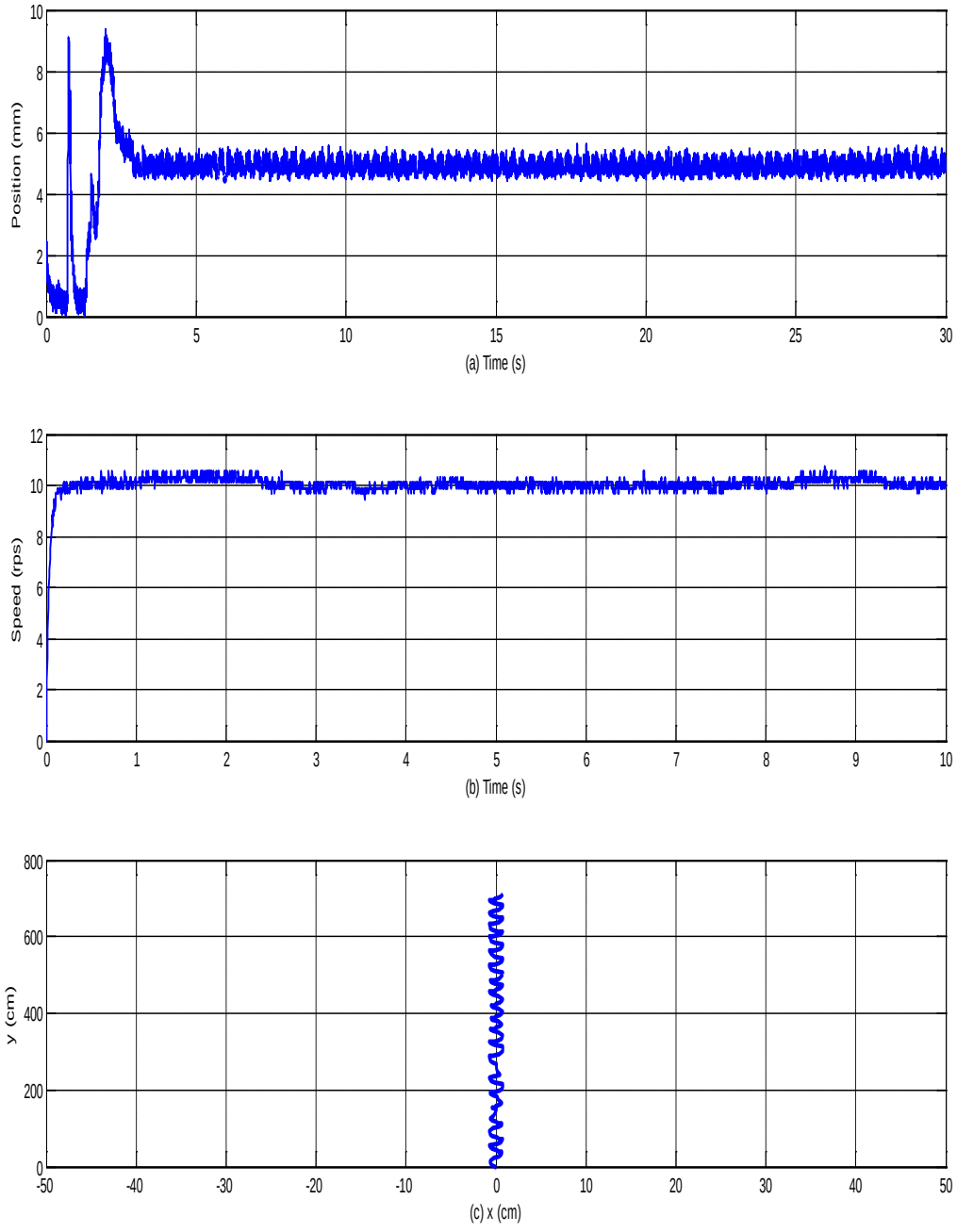


Fig. 32. Client motion trajectories from Case 1

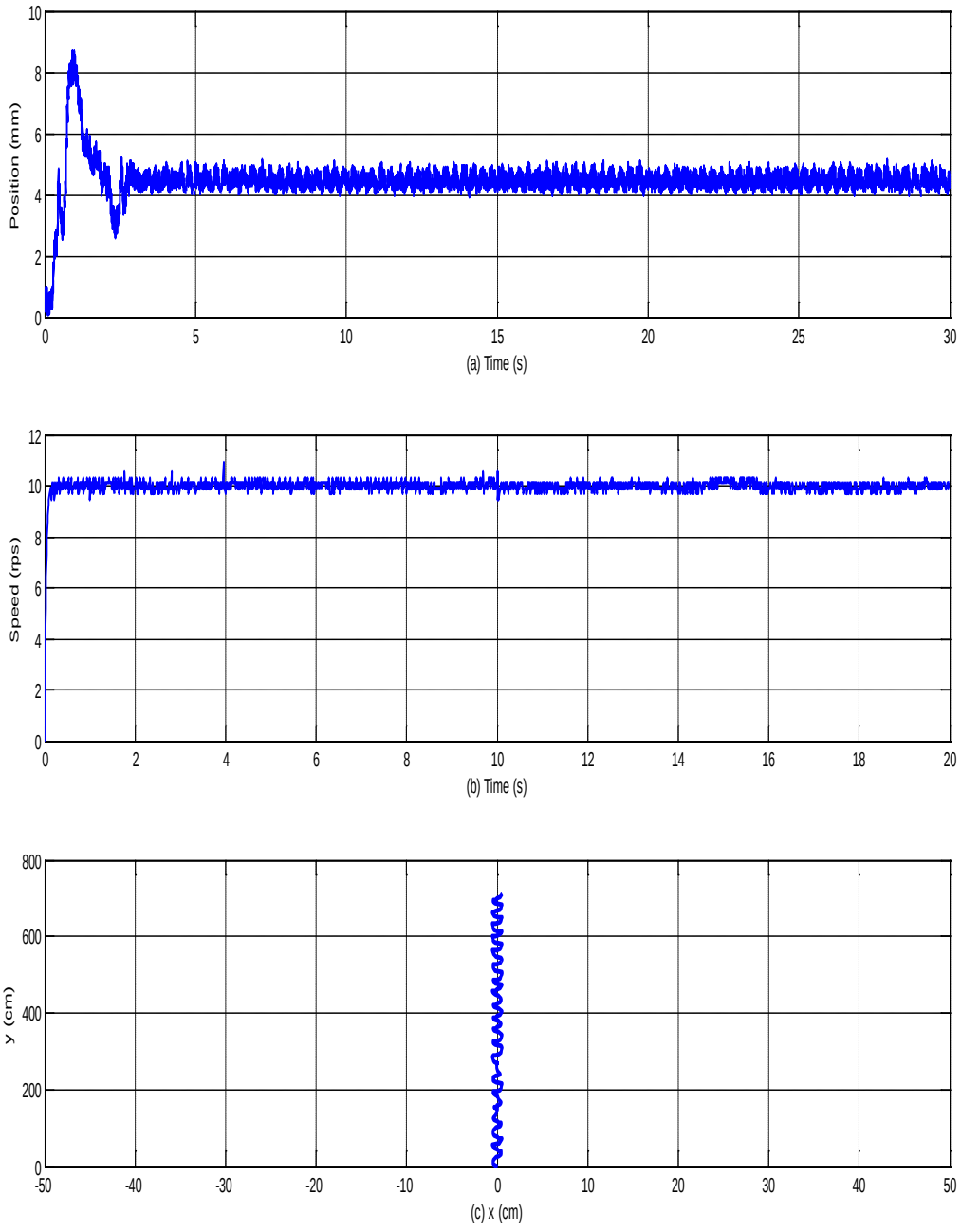


Fig. 33. Client motion trajectories from Case 2

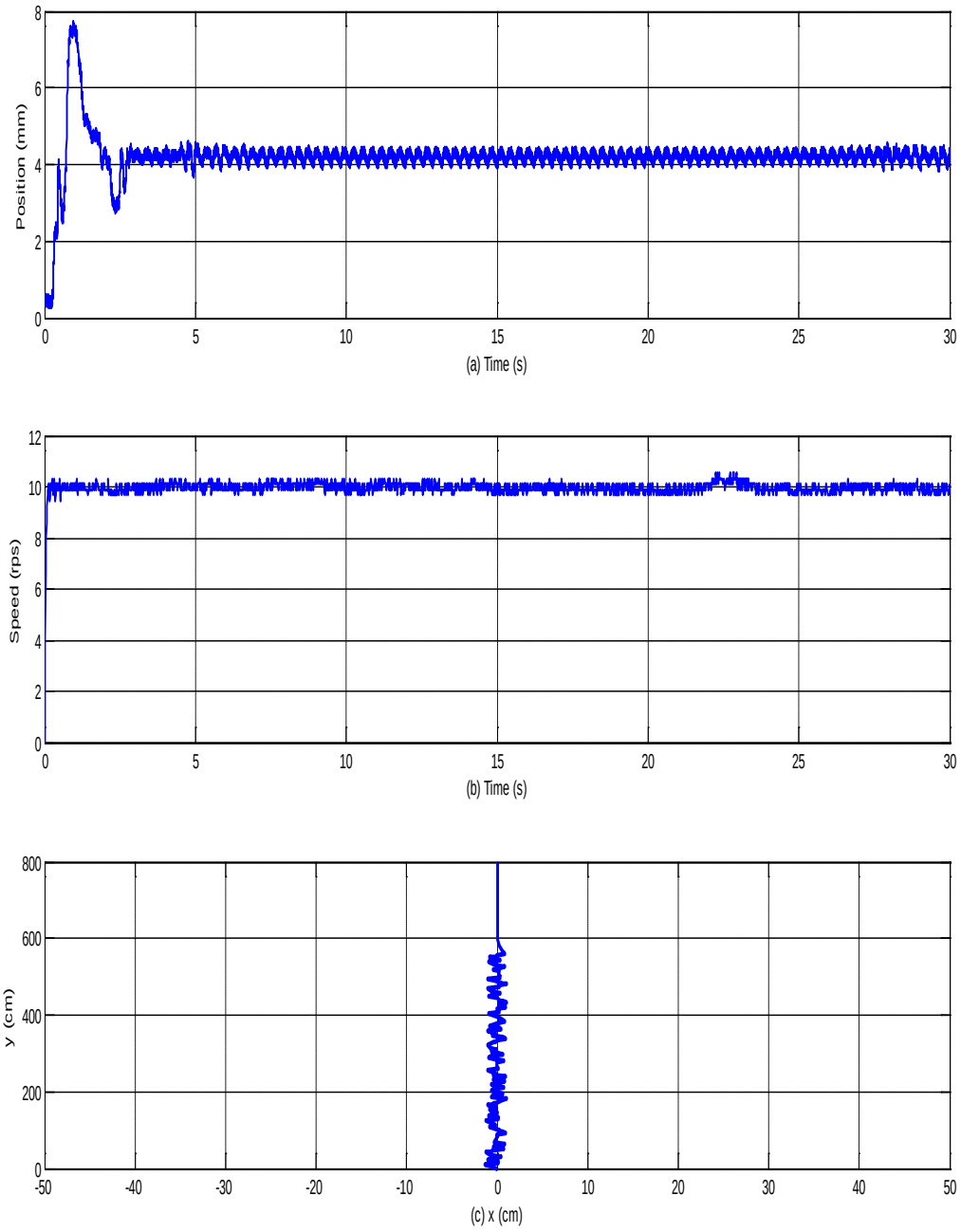


Fig. 34. Client motion trajectories from Case 3

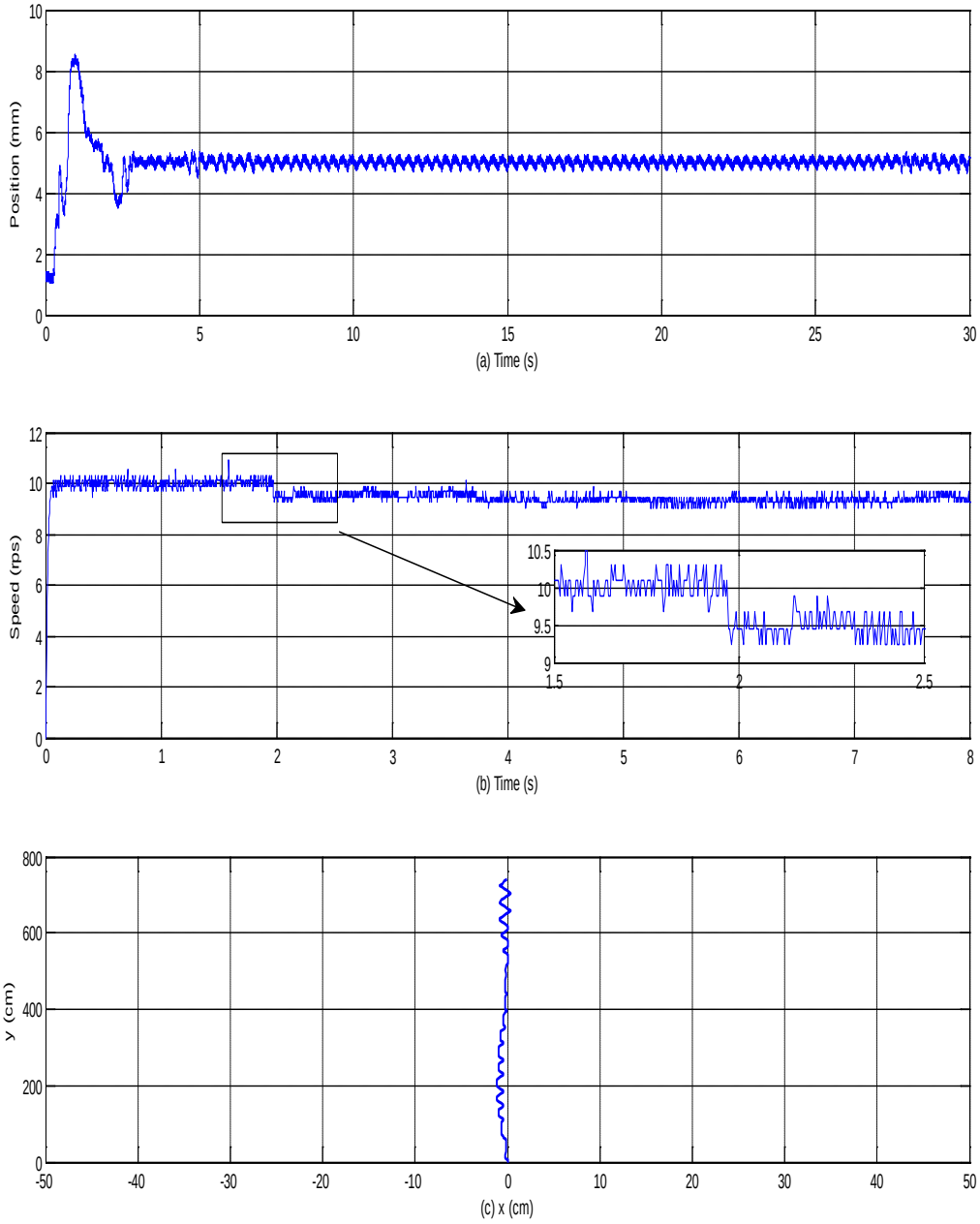


Fig. 35. Client motion trajectories from Case 4

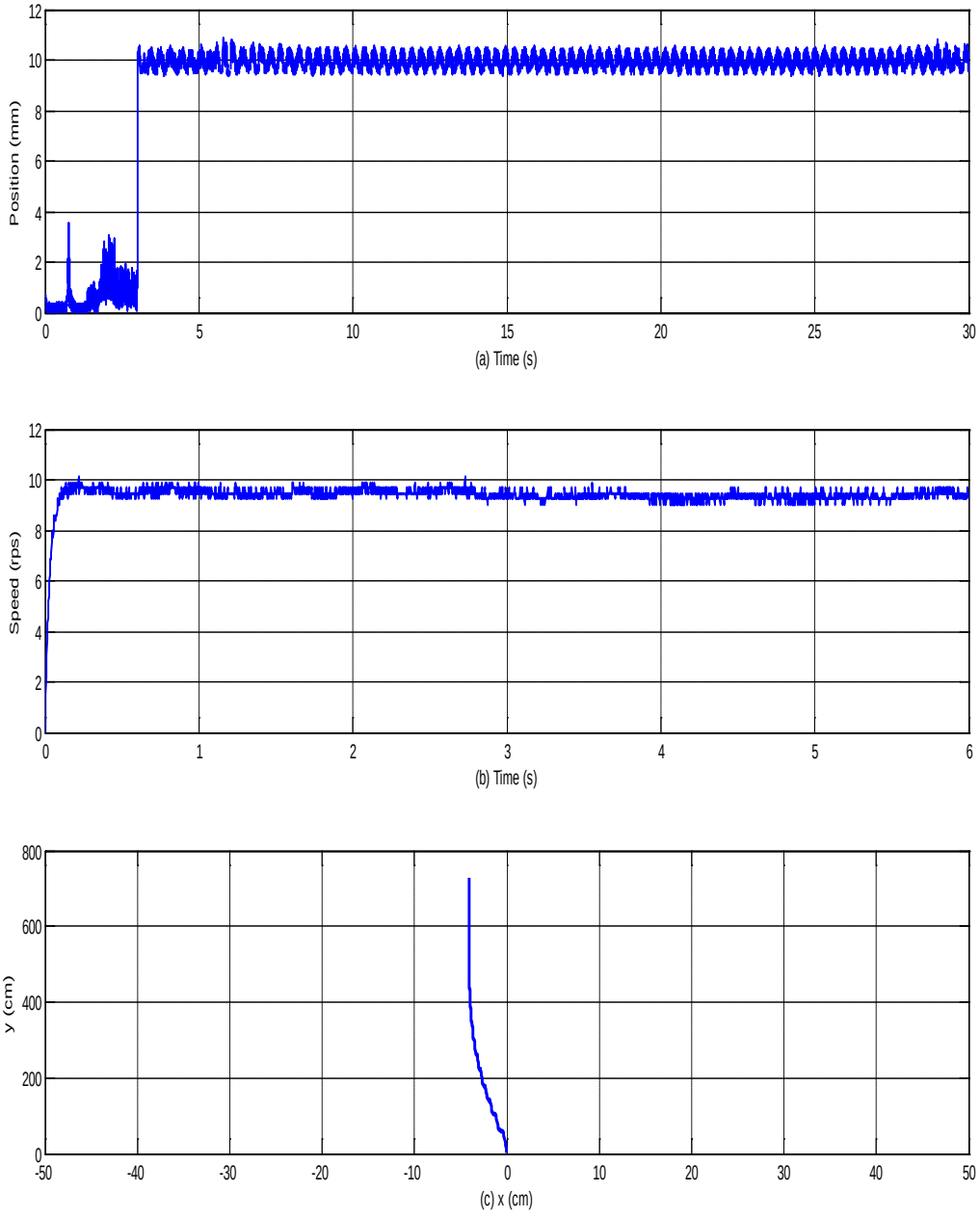


Fig. 36. Client motion trajectories from Case 4 without control flow

Table 5 shows the performance comparison of these four cases of the NCS. The numbers in Table 5 are means and standard deviations of the steady-state errors of each client. For Client 1, when the total BU increased by 10%, the average steady-state error increased by about 200%. For the other two clients, the medium client and the slow client, both the BU and the sampling periods affected the stability and the performance of the systems, but not as crucially as they were on the fast client, Client 1. If the TBU exceeds the BU threshold, the algorithm would bring the TBU of the NCS less than the threshold based on the type of each client's sampling frequency. From Table 5, the BU has more crucial effect on the system stability and performance as the dynamics of the system gets more complex.

Table 5. System performance comparisons of NCS with wireless client

Cases		1	2	3	4
Client 1 (mm)	Mean	0.7205	0.4851	0.2015	0.7653
	Stdev	0.4053	0.2860	0.2261	0.4158
Client 2 (rps)	Mean	0.0590	–0.0272	–0.0371	–0.0868
	Stdev	0.2010	0.1673	0.1745	0.1869
Client 3 (cm)	Mean	–0.0625	–0.0311	–0.1698	–0.0618
	Stdev	0.5529	0.2861	0.7320	0.4920

To determine the effects of the wireless client to the NCS, the experiments, cases 1 to 4, were performed with only the wired clients under the same BU as in Table 6.

Another DC motor speed-control system, Client 4, was introduced to replace the wireless robotic wheelchair, Client 3. Client 4 had exactly the same system configuration as that of Client 2. The execution time of Client 4 is 1.350 ms, assumed to be the same as Client 2. To maintain the same BU as in Table 5, Client 4's sampling periods were set as 7 ms, 10.7 ms, 21.4 ms, and 5.72 ms for Cases 1 to 4, respectively. Table 6 shows the performance comparison of these four cases of the NCS without wireless clients. The numbers in Table 6 follow the notations in Table 5. One would expect that Clients 2 and 4 have similar time responses if they are given the same sampling periods. Clients 2 and 4 have exactly the same system configuration and execution time, so Server will treat them equally. Although Clients 2 and 4 are identical plants, the data packets from each client may not arrive at Server at exactly the same time. In practice, however, the data packets will be queued up in Server's buffer waiting for unpacking and calculation. Different priorities can be assigned to the identical clients to rearrange their to-be-executed sequence. Although Clients 2 and 4 are identical in the experiments, their to-be-executed sequence can be different. Without modifications to the protocols, Clients 2 and 4 are served on the first-come first-serve base.

To show details of DIAE versus BU of Clients 2 and 4, separate experiments were conducted. Because of the uncertainties and the time delays on the network, five sets of experiments were conducted with 20,000 times for each given BU. Each experiment varied the BU of Clients 2 and 4 from 10% to 50%. The average of total DIAE of Clients 2 and 4 is shown in Fig. 37. From Fig. 37, the DIAEs of Clients 2 and 4 are nearly distributed evenly which verifies the earlier analysis.

Table 6. System performance comparisons of NCS without wireless client

Cases		1	2	3	4
Client 1 (mm)	Mean	0.5635	0.3858	–0.1286	0.7385
	Stdev	0.1402	0.1308	0.0916	0.1497
Client 2 (rps)	Mean	–0.4556	–0.4370	–0.2865	–0.4275
	Stdev	0.2269	0.2660	0.1969	0.2374
Client 4 (rps)	Mean	0.2178	0.2525	0.0617	0.6050
	Stdev	0.2569	0.2686	0.1913	0.2309

In the NCS without wireless clients, for the fast client, Client 1, with the same BU as in the NCS with wireless client, the average steady-state error decreased by about 20% to 30% compared with Table 6. Compared with Table 6, the wireless indeed introduced more complexity to the NCS with only wired clients. For the medium client, Clients 2, the average steady-state error increased because of the similar levels of the sampling periods as Client 4 although the BU was exactly the same. In this case, Clients 2 and 4 competed for the resources more fiercely compared to the NCS with Client 3.

From the analysis, the sampling period is not the only factor that will affect the stability and the performance of the each client in the NCS. The BU, the number of clients, and the structure of the network will determine the time-delay and packet-loss levels of the NCS, which will affect the stability and the performance of the each client. By Eq. (3), the sampling period and the BU are coupled parameters in the NCS. A large sampling period implies a smaller BU, thereby poor performance or even instability. A

small sampling period implies a larger BU, more time delays, or even packet losses. Therefore, the trade-off between the sampling period and the stability is necessary to control the NCS effectively.

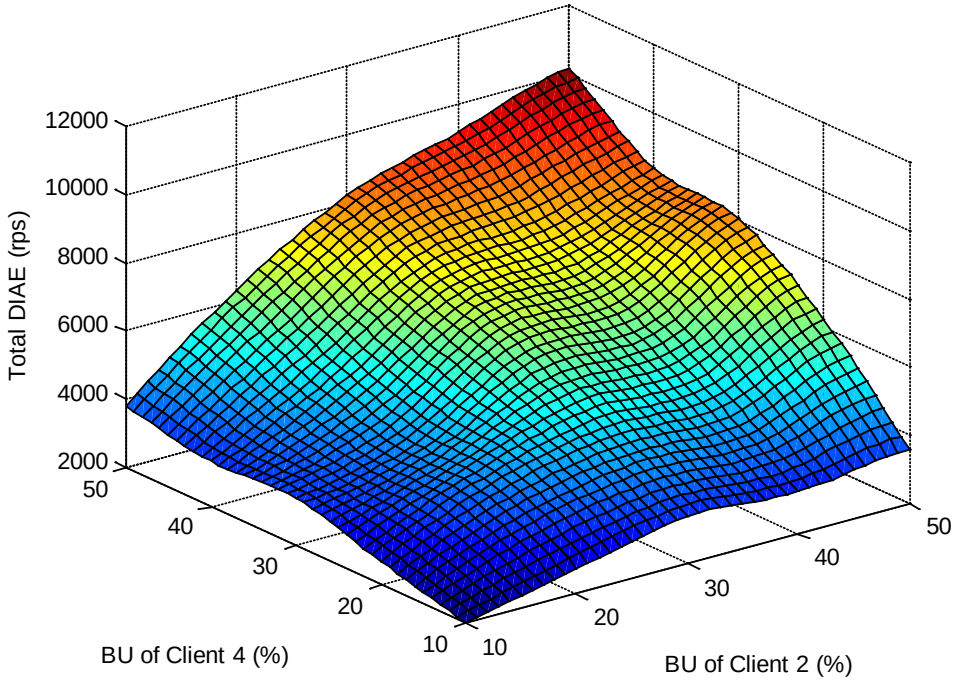


Fig. 37. Total DIAE vs. BUs of Clients 2 and 4

3.4 SUMMARY

This section discussed the fundamental elements of an NCS such as the time delays, packet losses, bandwidth allocation, etc. This section also presented a multiscale NCS that contains three different types of clients, defined as the fast, medium, and slow client, respectively. With the wireless capability brought by WLAN, the NCS expanded

its flexibility with the cost of complexity to its frame structure. This framework was adopted to test several control issues such as time delays, packet losses, and network bandwidth allocation of the NCS. The details of timing diagram, software and hardware setup of the NCS are illustrated. Several statistic and experimental results were given to verify the capability of the NCS.

4. MARKOV-CHAIN-BASED OUTPUT FEEDBACK CONTROL OF THE NCS*

The introduction of a communication network into a control system inevitably presents more constraints such as random time delays and packet losses that make the analysis and design of the NCS challenging. These random time delays and packet losses can degrade the system performance or even destabilize the system. How to compensate for the time delays and packet losses has become an active research area of the NCS.

In these aforementioned references [34–42], the authors assumed that the Markov-chain model could intuitively include the packet losses as well. However, the packet losses actually change the structure of the model. When a packet is lost, the sensor output or control input will be unavailable in all sense, whereas for the time-delay case, the sensor output or control input arrives at its destination node eventually with a certain amount of delays. Hence, the Markov-chain-based packet-loss model assumes that the packet-loss information can be included by the same probability transition from the time-delay perspective will not closely catch the nature of the NCS. Also in these aforementioned references, the stability analysis only considered the integer time-delay states. However, in the practical world, the time delays are non-integer numbers. In this dissertation, the random time delays and packet losses are treated with separate models

*Reprinted with permission from "Markov-chain-based output feedback control for stabilization of networked control systems with random time delays and packet losses" by J. Dong and W.-J. Kim, *International Journal of Control, Automation and Systems*, vol. 10, no. 5, pp. 1013–1322, Oct. 2012. Copyright 2012 by ICROS, KIEE and Springer.

that reveal the nature of the NCS in a closer manner. The proposed models for the time delays and packet losses are based on stochastic processes in the discrete-time domain so that the proposed method can be implemented on a practical NCS without much modification. The proposed method considers both integer and non-integer time delays.

4.1 SYSTEM MODELING

A typical NCS has a closed-loop structure as shown in Fig. 38. As indicated in the dashed boxes, Server represents the controller on one end of the communication network whereas Client represents the plant including sensors and actuators on the other end of the communication network.

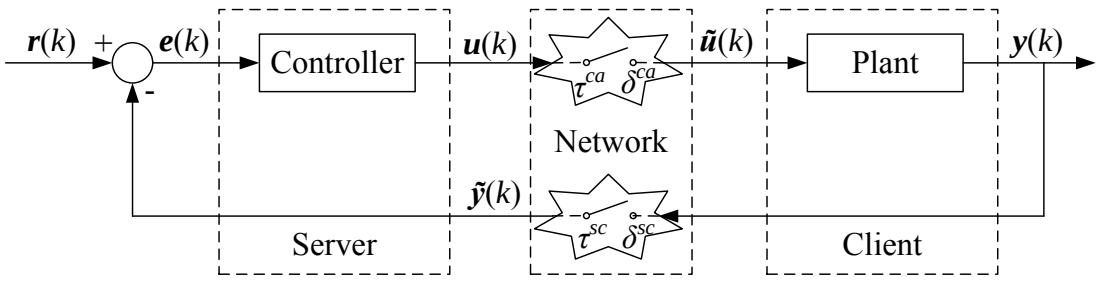


Fig. 38. A representative NCS block diagram

Assuming the entire NCS is a linear discrete-time system. τ^{ca} and τ^{sc} represent the random time delays, and δ^{ca} and δ^{sc}, the packet losses in the controller-to-actuator and sensor-to-controller links, respectively. The state-space model of the plant is

$$\boldsymbol{x}_p(k+1) = \boldsymbol{A}_p \boldsymbol{x}_p(k) + \boldsymbol{B}_p \tilde{\boldsymbol{u}}(k) \tag{15}$$

$$\boldsymbol{y}(k) = \boldsymbol{C}_p \boldsymbol{x}_p(k), \tag{16}$$

where $\boldsymbol{x}_p(k) \in \mathbb{R}^n$, $\boldsymbol{u}(k) \in \mathbb{R}^m$, and $\boldsymbol{y}(k) \in \mathbb{R}^p$ are state, control-input, and plant-output vectors, respectively. $\tilde{\boldsymbol{u}}(k) \in \mathbb{R}^m$ and $\tilde{\boldsymbol{y}}(k) \in \mathbb{R}^p$ are delayed control-input and plant-output vectors. $\boldsymbol{A}_p$, $\boldsymbol{B}_p$, and $\boldsymbol{C}_p$ are the known matrices with appropriated dimensions.

Similarly, the controller has a state-space model as

$$\boldsymbol{x}_c(k+1) = \boldsymbol{A}_c \boldsymbol{x}_c(k) + \boldsymbol{B}_c \boldsymbol{e}(k) \tag{17}$$

$$\boldsymbol{u}(k) = \boldsymbol{C}_c \boldsymbol{x}_c(k) + \boldsymbol{D}_c \boldsymbol{e}(k), \tag{18}$$

where $\boldsymbol{e}(k) = \boldsymbol{r}(k) - \tilde{\boldsymbol{y}}(k)$ is error and $\boldsymbol{r}(k) \in \mathbb{R}^p$ is reference command. $\boldsymbol{A}_c$, $\boldsymbol{B}_c$, $\boldsymbol{C}_c$, and $\boldsymbol{D}_c$ are to be determined to compensate for the random time delays and packet losses, which will be discussed in Section 4.1.4 with details that include the stability criterion and algorithm. An experimental example of how to design the controller matrices will be given in Section 3.3.

4.1.1. Markov Chain

A Markov chain is a mathematical system that undergoes transitions from one state to another that belongs to a set of finite or countable number of possible states. It is a random process characterized as memoryless so that the next state depends only on the current state and not on the sequence of events that preceded it.

Let Ψ be a sample space that contains finite states. Consider a stochastic process $X = \{X_n;\ n \in \mathrm{N} = 0,\ 1,\ \cdots\}$ within the countable state space Ψ. Then "the process is at

state i at time n" means that $X_n = i,\ i \in \Psi,\ n \in \mathrm{N}$. The definition of a Markov chain is then given as follows.

Definition 1 [72]: The stochastic process $X = \{X_n;\ n \in \mathrm{N} = 0,\ 1,\ \cdots\}$ is called a Markov chain provided that $\Pr\{X_{n+1} = i \mid X_0,\ X_1,\ \cdots,\ X_n\} = \Pr\{X_{n+1} = i \mid X_n\}$ for all $i \in \Psi$ and $n \in \mathrm{N}$. A Markov chain is then a sequence of random variables so that the next state X_{n+1} of the process is independent of the past states $X_0,\ X_1,\ \cdots,\ X_{n-1}$ provided that the present state X_n is known.

Definition 2 [72]: The probabilities $\Pr(i,\ j)$ are called the transition probabilities for the Markov chain X with $\Pr\{X_{n+1} = j \mid X_n = i\} = \Pr(i,\ j)$ where $i,\ j \in \Psi$. And a Markov chain X satisfying this definition is said to be time-homogeneous if emphasis is needed.

Definition 3 [72]: The transition-probability matrix of a Markov chain X is

$$TP = \begin{bmatrix} \Pr(0,0) & \Pr(0,1) & \Pr(0,2) & \cdots \\ \Pr(1,0) & \Pr(1,1) & \Pr(1,2) & \cdots \\ \Pr(2,0) & \Pr(2,1) & \Pr(2,2) & \cdots \\ \vdots & \vdots & \vdots & \cdots \end{bmatrix}$$

if $\Psi = \{0,\ 1,\ \cdots\}$.

Figure 39 shows an example of the Markov Chain with its transition-probability matrix. This Markov chain contains three states $\Psi = \{0,\ 1,\ 2\}$ and has a transition-probability matrix $TP = \begin{bmatrix} 0.5 & 0.25 & 0.25 \\ 0.025 & 0.9 & 0.075 \\ 0.05 & 0.15 & 0.8 \end{bmatrix}$.

4.1.2. Time-Delay Modeling

In general, time delays can be categorized as deterministic delays and stochastic delays. Due to the stochastic nature of the network, a stochastic method is adopted to model the random time delays in the communication links since it can model the random processes of the network condition more realistically compared to a deterministic method. We assume that the status of the time delays mainly depends on the previous status so that the random time delays τ^{ca} and τ^{sc} can be modeled with finite-state time-homogeneous Markov chains.

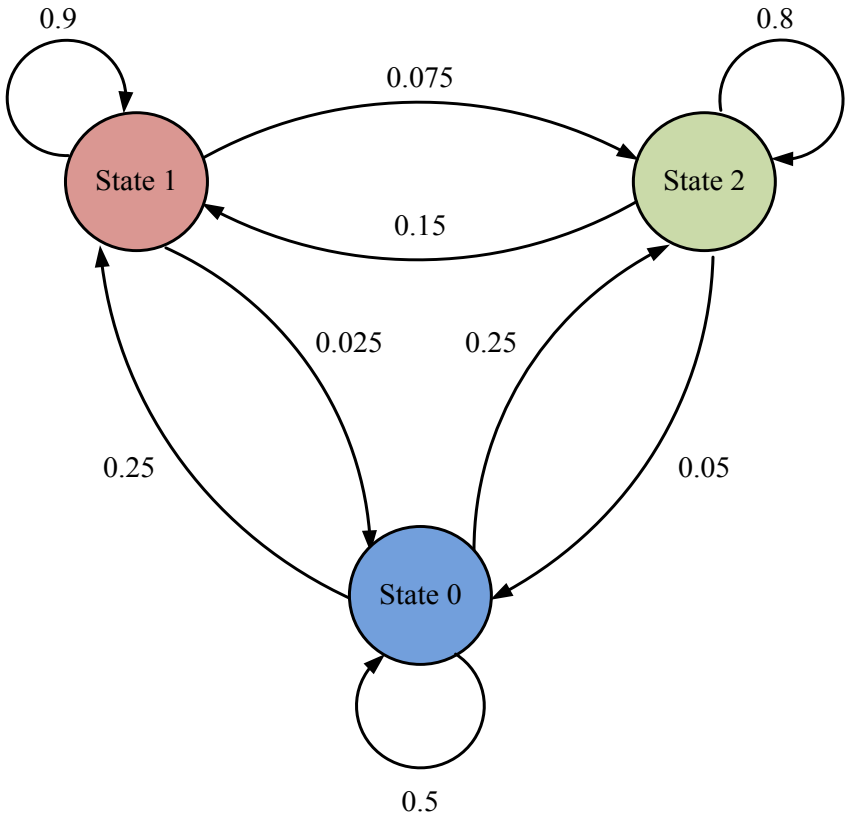


Fig. 39. An example of Markov Chain with three states

In this section, τ^{ca} and τ^{sc} in Fig. 38 are modeled with two time-homogeneous Markov chains with finite Markov states and take values in the sets $\mathbb{P} = \{\tau_i^{ca};\ i \in \mathbb{I} = 1,\ \cdots,\ p\}$ and $\mathbb{Q} = \{\tau_m^{sc};\ m \in \mathbb{M} = 1,\ \cdots,\ q\}$, respectively. Their transition-probability matrices are $\boldsymbol{\Lambda} = \{\varsigma_{ij}\}$ and $\boldsymbol{\Gamma} = \{\xi_{mn}\}$, respectively. These transition-probability matrices represent the probabilities that τ^{ca} and τ^{sc} jump from the state *i* to *j* and the state *m* to *n*, respectively. The definitions of ς_{ij} and ξ_{mn} are

$$\varsigma_{ij} = \Pr(\tau^{ca}(k+1) = \tau_j^{ca} \mid \tau^{ca}(k) = \tau_i^{ca}), \tag{19}$$

$$\xi_{mn} = \Pr(\tau^{sc}(k+1) = \tau_n^{sc} \mid \tau^{sc}(k) = \tau_m^{sc}), \tag{20}$$

where $\varsigma_{ij} \geq 0$, $\xi_{mn} \geq 0$, and $\sum_{j=1}^{p} \varsigma_{ij} = 1$, $\sum_{n=1}^{q} \xi_{mn} = 1$, for all $i, j \in \mathbb{I}$ and $m, n \in \mathbb{M}$.

4.1.3. Packet-Loss Modeling

The packet loss in an NCS is another challenge induced by the network. Packet losses could take place when the network is congested or the queues of routers and servers are overflown. NCS does not monitor the network conditions, so explicit packet-loss information is unavailable to either Server or Client in the sense of real time.

A simplest stochastic model treats packet losses as a Bernoulli process [5]. It can also be modeled with a Markov chain [73] or a Poisson process [74]. Normally, packet losses share no common probabilistic characteristics with the random time delays since their causes are usually different and not always coupled. In general, for the case of packet losses, the system will require extensive control input to guarantee the stability and system performance. Whenever a packet is lost, time-delay information is irrelevant

and unavailable. Therefore, assuming that a packet loss can be intuitively modeled with a Markov chain together with the time delays cannot represent their independence in a communication network. In this section, a separate packet-loss model is introduced.

As illustrated in Fig. 38, the network backbone can be treated as a jump system. In this case, when a packet is lost, the current output packet or the control input packet will be unavailable to either Server or Client, so that the plant output or control input from the previous sampling period will be held for current period. The time-delay information from the previous period will also be inherited.

The notations of the packet losses in Fig. 38 are as below.

$$\delta^{ca}(k) = \begin{cases} 1 & \text{if no packet is lost} \\ 0 & \text{if a packet is lost} \end{cases} \tag{21}$$

$$\delta^{sc}(k) = \begin{cases} 1 & \text{if no packet is lost} \\ 0 & \text{if a packet is lost} \end{cases} \tag{22}$$

Unlike [5], however, we do not assign Bernoulli probabilities to δ^{ca} and δ^{sc}. Packet losses can be stochastic so that a pre-assigned fixed probability $\Pr(\delta^{ca}(k)=1)$ would not represent the nature of the packet losses realistically. That is, if either δ^{ca} and δ^{sc} takes the value 0, there is a packet lost in the corresponding links. Otherwise, only random time delays exist in the links.

4.1.4. Controller Design

As in Fig. 40, consider τ^{ca} and τ^{sc}, we introduce a ceiling function

$$f(\tau) = \left\lceil \frac{\tau + \tau_0}{h} \right\rceil, \tag{23}$$

where τ_0 is time threshold, and h is the sampling period. The time threshold τ_0 includes summation of the data sampling time, data-packet generating time, packet-processing time, queuing time, etc. In each sampling period, these times may not be exactly the same, but can be quite deterministic. Therefore, an upper bound τ_0 can be set as a time threshold. Then the plant-output packet arriving at Server is $\tilde{\boldsymbol{y}}(k) = \boldsymbol{y}(k - f(\tau^{sc}(k)))$. This can also be applied to the control input, so that $\tilde{\boldsymbol{u}}(k) = \boldsymbol{u}(k - f(\tau^{ca}(k)))$. Note that for τ^{ca} and τ^{sc}, the threshold τ_0 may take different values.

Figure 40 illustrates an example of packet exchanges between Server and Client. The horizontal length of each line indicates the random time delays of each packet in the links. Several possible scenarios are shown in Fig. 40. The first case is that both the τ^{ca} and τ^{sc} are shorter than h as shown in h_1. Another case is that τ^{sc} is shorter than h and τ^{ca} is longer than h as shown in h_2. For instance, if $\tau^{sc} + \tau_0 < h$, then $f(\tau^{sc}) = 0$. Thus, when the plant-output packet arrives at Server, it is indicated as $\tilde{\boldsymbol{y}}(k)$ in the k-th sampling period. Likewise, if $h < \tau^{sc} + \tau_0 < 2h$, then $f(\tau^{sc}) = 1$. Thus the plant-output packet arrives at Server will be $\tilde{\boldsymbol{y}}(k-1)$ in the k-th sampling period. Figure 40 also illustrates other possible data-packet-loss scenarios. As shown in h_3 and h_5, the plant outputs and the control inputs are lost in data transmission, respectively. Hence, the estimated or predicted data will be applied to calculate the corresponding data packets.

Now consider the NCS in Fig. 38 with both the random time delays and the packet losses. The delayed plant outputs $\tilde{\boldsymbol{y}}(k)$ and control inputs $\tilde{\boldsymbol{u}}(k)$ are

$$\tilde{\boldsymbol{y}}(k) = \delta^{sc}(k)\boldsymbol{y}(k - f(\tau^{sc}(k))) + \bar{\delta}^{sc}(k)\boldsymbol{y}(k-1- f(\tau^{sc}(k-1))) \tag{24}$$

$$\tilde{\boldsymbol{u}}(k) = \delta^{ca}(k)\boldsymbol{u}(k - f(\tau^{ca}(k))) + \bar{\delta}^{ca}(k)\boldsymbol{u}(k-1- f(\tau^{ca}(k-1))), \tag{25}$$

where $\bar{\delta}^{sc}(k) = 1 - \delta^{sc}(k)$ and $\bar{\delta}^{ca}(k) = 1 - \delta^{ca}(k)$. By Eqs. (24) and (25), if packet losses take place in the links, the previous data packets will be used. This provision can compensate for one packet loss.

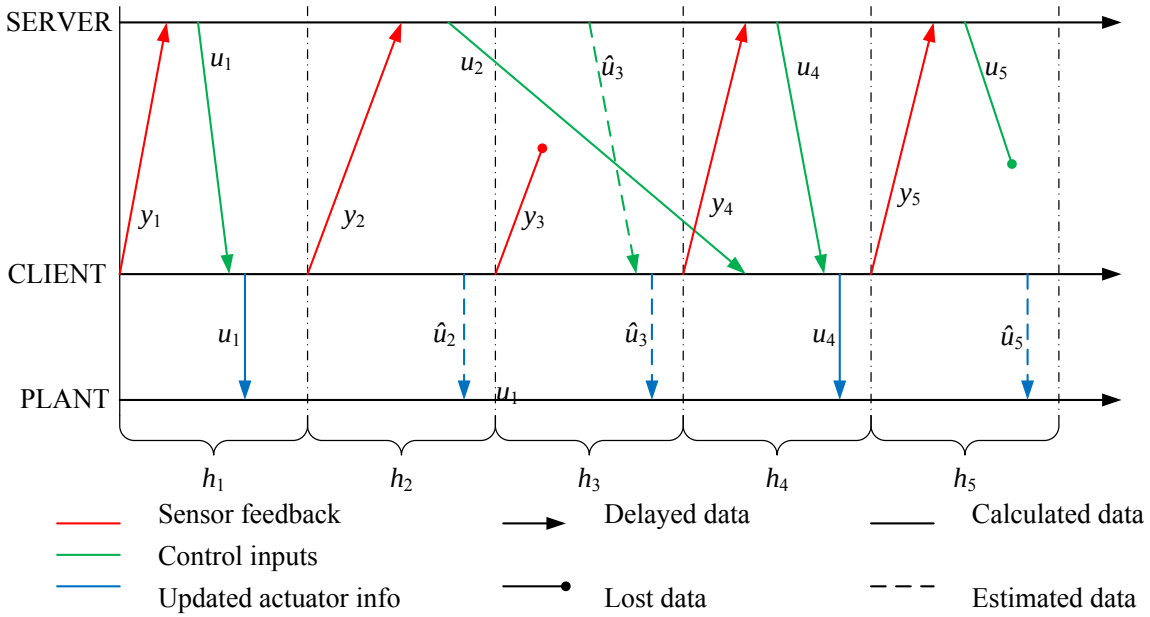


Fig. 40. An example timing diagram of the NCS communication

For consecutive packets losses, an autoregressive (AR) model will be applied to predict future plant outputs and control inputs of an NCS.

Definition 4 [75]: A simple input-output relationship is obtained by describing it as a linear difference equation:

$$y(t)+a_1y(t-1)+\cdots+a_{n_a}y(t-n_a)=b_1u(t-1)+\cdots+b_{n_b}u(t-n_b),$$

or

$$A(q)y(t)=B(q)u(t),$$

where $A(q)=1+a_1q^{-1}+a_2q^{-2}+\cdots+a_{n_a}q^{-n_a}$ and $B(q)=b_1q^{-1}+b_2q^{-2}+\cdots+b_{n_b}q^{-n_b}$, n_a is the order of $A(q)$ and n_b is the order of $B(q)$. The above model is called an autoregressive exogenous (ARX) model, where AR refers to the autoregressive part $A(q)y(t)$ and X to the extra exogenous input $B(q)u(t)$.

The AR model will be applied to predict the future data packets. The extra input term is dropped off from the ARX model bacause when the packet losses exist in the network, the input information is unavilable to its destination node. Therefore, to either Sever or Client, the plant outputs or control inputs are unavilable in the present of packet losses. Then the plant outputs to the controller $\tilde{\boldsymbol{y}}(k)$ and the control inputs to the plant $\tilde{\boldsymbol{u}}(k)$ are as follows for the case of consecutive packets losses

$$\tilde{\boldsymbol{y}}(k)=\delta^{sc}(k)\boldsymbol{y}(k-f(\tau^{sc}(k)))+\bar{\delta}^{sc}(k)\hat{\boldsymbol{y}}(k) \tag{26}$$

$$\tilde{\boldsymbol{u}}(k)=\delta^{ca}(k)\boldsymbol{u}(k-f(\tau^{ca}(k)))+\bar{\delta}^{ca}(k)\hat{\boldsymbol{u}}(k), \tag{27}$$

where $\hat{\boldsymbol{y}}(k)$ and $\hat{\boldsymbol{u}}(k)$ are the predicted data packets generated by the AR model with

$$\begin{aligned}\hat{\boldsymbol{y}}(k)&=(-a_{y1}q^{-1}-a_{y2}q^{-2}-\cdots-a_{yn_a}q^{-n_a})\boldsymbol{y}(k-f(\tau^{sc}(k)))\\&=-a_{y1}\boldsymbol{y}(k-1-f(\tau^{sc}(k-1)))-\cdots-a_{yn_a}\boldsymbol{y}(k-yn_a-f(\tau^{sc}(k-yn_a))),\end{aligned} \tag{28}$$

$$\begin{aligned}\hat{\boldsymbol{u}}(k)&=(-a_{u1}q^{-1}-a_{u2}q^{-2}-\cdots-a_{un_a}q^{-n_a})\boldsymbol{u}(k-f(\tau^{ca}(k)))\\&=-a_{u1}\boldsymbol{u}(k-1-f(\tau^{ca}(k-1)))-\cdots-a_{un_a}\boldsymbol{u}(k-un_a-f(\tau^{ca}(k-un_a))),\end{aligned} \tag{29}$$

where yn_a and un_a are orders of the predicted plant outputs and control inputs, respectively. Based on the time-delay states of the NCS, we have $yn_a \le q$ and $un_a \le p$.

Augment the plant's states as follows with all the possible Markov states of τ^{sc}

$$\bar{\boldsymbol{x}}_p(k) = \left[\boldsymbol{x}_p^T(k) \quad \boldsymbol{y}^T(k-1) \quad \boldsymbol{y}^T(k-2) \quad \cdots \quad \boldsymbol{y}^T(k-q-1) \right]^T.$$

Then the plant's model can be written as

$$\bar{\boldsymbol{x}}_p(k+1) = \bar{\boldsymbol{A}}_p \bar{\boldsymbol{x}}_p(k) + \bar{\boldsymbol{B}}_p \tilde{\boldsymbol{u}}(k) \tag{30}$$

$$\tilde{\boldsymbol{y}}(k) = \bar{\boldsymbol{C}}_p \bar{\boldsymbol{x}}_p(k), \tag{31}$$

where $\bar{\boldsymbol{A}}_p = \begin{bmatrix} \boldsymbol{A}_p & \mathbf{0} & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ \boldsymbol{C}_p & \mathbf{0} & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ddots & \cdots & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \cdots & \boldsymbol{I} & \mathbf{0} \end{bmatrix}$, $\bar{\boldsymbol{B}}_p = \begin{bmatrix} \boldsymbol{B}_p \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}$,

$\bar{\boldsymbol{C}}_p = \left[\mathbf{0} \quad \cdots \quad \delta^{sc}(k)\mathbf{1} \quad \bar{\delta}^{sc}(k)\mathbf{1} \quad \cdots \quad \mathbf{0} \right]$ with the $f(\tau^{sc}(k))$-th entry equals to $\delta^{sc}(k)\mathbf{1}$

where $\mathbf{1}$ is the unit matrix with all elements equal to 1. For the case of consecutive packet losses, $\bar{\boldsymbol{C}}_p = \left[\mathbf{0} \quad \cdots \quad \delta^{sc}(k)\mathbf{1} \quad \underbrace{-a_{y1}\bar{\delta}^{sc}(k)\mathbf{1} \quad \cdots \quad -a_{yn_a}\bar{\delta}^{sc}(k)\mathbf{1}}_{yn_a} \quad \cdots \quad \mathbf{0} \right]$.

Similarly, the augmented controller state vector with all the possible Markov states of τ^{ca} is $\bar{\boldsymbol{x}}_c(k) = \left[\boldsymbol{x}_c^T(k) \quad \boldsymbol{u}^T(k-1) \quad \boldsymbol{u}^T(k-2) \quad \cdots \quad \boldsymbol{u}^T(k-p-1) \right]^T$, and the corresponding controller model is

$$\bar{\boldsymbol{x}}_c(k+1) = \bar{\boldsymbol{A}}_c \bar{\boldsymbol{x}}_c(k) - \bar{\boldsymbol{B}}_c \tilde{\boldsymbol{y}}(k) + \bar{\boldsymbol{B}}_c \boldsymbol{r}(k) \tag{32}$$

$$\tilde{\boldsymbol{u}}(k) = \bar{\boldsymbol{C}}_c \bar{\boldsymbol{x}}_c(k), \tag{33}$$

where $\bar{\boldsymbol{A}}_c = \begin{bmatrix} \boldsymbol{A}_c & \boldsymbol{0} & \boldsymbol{0} & \cdots & \cdots & \boldsymbol{0} \\ \boldsymbol{C}_c & \boldsymbol{0} & \boldsymbol{0} & \cdots & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} & \cdots & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \ddots & \cdots & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \cdots & \boldsymbol{I} & \boldsymbol{0} \end{bmatrix}$, $\bar{\boldsymbol{B}}_c = \begin{bmatrix} \boldsymbol{B}_c \\ \boldsymbol{D}_c \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \end{bmatrix}$,

$\bar{\boldsymbol{C}}_c = \begin{bmatrix} \boldsymbol{0} & \cdots & \delta^{ca}(k)\boldsymbol{1} & \bar{\delta}^{cc}(k)\boldsymbol{1} & \cdots & \boldsymbol{0} \end{bmatrix}$ with the $f(\tau^{ca}(k))$-th entry equals to $\delta^{ca}(k)\boldsymbol{1}$.

Similarly, for the case of consecutive packets losses,

$$\bar{\boldsymbol{C}}_c = \begin{bmatrix} \boldsymbol{0} & \cdots & \delta^{ca}(k)\boldsymbol{1} & \underbrace{-a_{u1}\bar{\delta}^{ca}(k)\boldsymbol{1} \quad \cdots \quad -a_{un_a}\bar{\delta}^{ca}(k)\boldsymbol{1}}_{un_a} & \cdots & \boldsymbol{0} \end{bmatrix}.$$

Augment the new plant and controller model with $\bar{\boldsymbol{x}} = \begin{bmatrix} \bar{\boldsymbol{x}}_p^T & \bar{\boldsymbol{x}}_c^T \end{bmatrix}^T$, and the closed-loop dynamics will be

$$\bar{\boldsymbol{x}}(k+1) = (\boldsymbol{A} + \boldsymbol{BKC})\bar{\boldsymbol{x}}(k), \tag{34}$$

where $\boldsymbol{A} = \begin{bmatrix} \bar{\boldsymbol{A}}_p & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}$, $\boldsymbol{B} = \begin{bmatrix} \boldsymbol{0} & \bar{\boldsymbol{B}}_p \\ \boldsymbol{I} & \boldsymbol{0} \end{bmatrix}$, $\boldsymbol{K} = \begin{bmatrix} \bar{\boldsymbol{A}}_c & -\bar{\boldsymbol{B}}_c \\ \bar{\boldsymbol{C}}_c & \boldsymbol{0} \end{bmatrix}$, and $\boldsymbol{C} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ \bar{\boldsymbol{C}}_p & \boldsymbol{0} \end{bmatrix}$.

For the stability analysis, *Definition 4* in [76] and *Theorem 1* in [77] are adopted.

Definition 5 [76]: Consider a jump linear system J_d

$$J_d : \begin{cases} \boldsymbol{x}(k+1) = \boldsymbol{A}(\eta(k))\boldsymbol{x}(k) + \boldsymbol{B}(\eta(k))\boldsymbol{u}(k) \\ \boldsymbol{y}(k) = \boldsymbol{C}(\eta(k))\boldsymbol{x}(k) + \boldsymbol{D}(\eta(k))\boldsymbol{u}(k) \end{cases},$$

where $\eta(k)$ is a discrete homogeneous Markov chain with states $S = (S_1,\ S_2,\ \cdots,\ S_N)$. The system J_d with $\boldsymbol{u}(k) = \boldsymbol{0}$ is said to be asymptotic mean-square stable if

$$E\{\|\bar{\boldsymbol{x}}(k)\|^2\} \to 0 \text{ as } k \to \infty,$$

for any initial condition $\boldsymbol{x}(0) = \boldsymbol{x}_0$ and initial distribution $\eta(0) = \eta_0$.

Theorem 1 [77]: Let there exist a nonnegative functional $V_i = V(i, x_{-h}, \cdots, x_i)$, $i \in \mathbb{Z}$ for which the conditions

$$E\{\Delta V_i\} \le -cE\{x_i^2\},$$

where $\Delta V_i = V_{i+1} - V_i$ and $c > 0$ hold. Then system J_d is asymptotic mean-square stable.

With *Theorem 1*, the necessary and sufficient conditions of the asymptotic mean-square stability of the closed-loop system Eq. (34) can be derived as follows.

Theorem 2: The closed-loop NCS in Eq. (34) is asymptotic mean-square stable if and only if there exists $\boldsymbol{P}(i,m) = \boldsymbol{P}^T(i,m) > 0$ so that the following matrix inequality

$$\boldsymbol{H}(i,m) = (\boldsymbol{A} + \boldsymbol{BKC})^T (\sum_{j=1}^{p} \sum_{n=1}^{q} \lambda_{ij} \mu_{mn} \boldsymbol{P}(j,n))(\boldsymbol{A} + \boldsymbol{BKC}) - \boldsymbol{P}(i,m) < 0 \tag{35}$$

holds for all $i \in \mathbb{I}$ and $m \in \mathbb{M}$.

Proof: Sufficiency: for the closed-loop NCS in Eq. (34), construct a Lyapunov function as

$$V(\bar{\boldsymbol{x}}(k), k) = \bar{\boldsymbol{x}}^T(k) \boldsymbol{P}(\tau^{ca}(k), \tau^{sc}(k)) \bar{\boldsymbol{x}}(k). \tag{36}$$

Then

$$\begin{aligned} \Delta V(\bar{\boldsymbol{x}}(k), k) &= V(\bar{\boldsymbol{x}}(k+1), k+1) - V(\bar{\boldsymbol{x}}(k), k) \\ &= \bar{\boldsymbol{x}}^T(k+1) \boldsymbol{P}(\tau^{ca}(k+1), \tau^{sc}(k+1)) \bar{\boldsymbol{x}}(k+1) - \bar{\boldsymbol{x}}^T(k) \boldsymbol{P}(\tau^{ca}(k), \tau^{sc}(k)) \bar{\boldsymbol{x}}(k). \end{aligned} \tag{37}$$

Assume τ^{ca} is at the Markov state *i* in the *k*-th sampling period and will be at the Markov state *j* in the next sampling period. Similarly, τ^{sc} is at the Markov state *m* in the *k*-th sampling period and will be at the Markov state *n* in the next sampling period. For the simplicity, we denote $\boldsymbol{P}(\tau^{ca}(k), \tau^{sc}(k))$ as $\boldsymbol{P}(i,m)$ hereafter. Note that (i,m) is not the corresponding entry of matrix $\boldsymbol{P}$, but the corresponding Markov states of the time delays.

Then Eq. (37) can be reformulated as

$$\begin{aligned}\Delta V(\bar{\boldsymbol{x}}(k),k) &= \bar{\boldsymbol{x}}^T(k+1)\boldsymbol{P}(j,n)\bar{\boldsymbol{x}}(k+1) - \bar{\boldsymbol{x}}^T(k)\boldsymbol{P}(i,m)\bar{\boldsymbol{x}}(k) \\ &= \bar{\boldsymbol{x}}^T(k)[(\boldsymbol{A}+\boldsymbol{BKC})^T\boldsymbol{P}(j,n)(\boldsymbol{A}+\boldsymbol{BKC}) - \boldsymbol{P}(i,m)]\bar{\boldsymbol{x}}(k). \end{aligned} \tag{38}$$

The next time-delay state will depend on the current one, and the conditional expectation of Eq. (38) is as follows.

$$\begin{aligned}E\{\Delta V(\bar{\boldsymbol{x}}(k),k)\} &= E\{\bar{\boldsymbol{x}}^T(k)[(\boldsymbol{A}+\boldsymbol{BKC})^T(\boldsymbol{P}(j,n)\,|\,\boldsymbol{P}(i,m))(\boldsymbol{A}+\boldsymbol{BKC}) - \boldsymbol{P}(i,m)]\bar{\boldsymbol{x}}(k)\} \\ &= E\{\bar{\boldsymbol{x}}^T(k)(\boldsymbol{A}+\boldsymbol{BKC})^T(\sum_{j=1}^{p}\sum_{n=1}^{q}\lambda_{ij}\mu_{mn}\boldsymbol{P}(j,n))(\boldsymbol{A}+\boldsymbol{BKC}) - \boldsymbol{P}(i,m)\bar{\boldsymbol{x}}(k)\} \\ &= E\{\bar{\boldsymbol{x}}^T(k)\boldsymbol{H}(i,m)\bar{\boldsymbol{x}}(k)\}. \end{aligned} \tag{39}$$

If $\boldsymbol{H}(i,m) < 0$, then

$$\begin{aligned}E\{\Delta V(\bar{\boldsymbol{x}}(k),k)\} &= E\{\bar{\boldsymbol{x}}^T(k)\boldsymbol{H}(i,m)\bar{\boldsymbol{x}}(k)\} \\ &\le E\{-\sigma_{\min}(i,m)\bar{\boldsymbol{x}}^T(k)\bar{\boldsymbol{x}}(k)\} \\ &\le -\sigma E\{\|\bar{\boldsymbol{x}}(k)\|^2\}, \end{aligned} \tag{40}$$

where $\sigma_{\min}(i,m) = \sigma_{\min}(-\boldsymbol{H}(i,m))$ is the minimum eigenvalue of $-\boldsymbol{H}(i,m)$ and $\sigma = \inf\{\sigma_{\min}(i,m),\ i \in \mathbb{I},\ m \in \mathbb{M}\} > 0$ is the infimum of these minimum eigenvalues.

According to Theorem 1, if $\boldsymbol{H}(i,m) < 0$, then the closed-loop system Eq. (34) is asymptotic mean-square stable.

Necessity: Under the assumption that Eq. (34) is asymptotic mean-square stable and *Theorem* 1, with a constant $\alpha > 0$, one has

$$\begin{aligned} E\{\Delta V(\bar{\boldsymbol{x}}(k),k)\} &= E\{\bar{\boldsymbol{x}}^T(k)\boldsymbol{H}(i,m)\bar{\boldsymbol{x}}(k)\} \\ &\le -\alpha E\{\|\bar{\boldsymbol{x}}(k)\|^2\} \\ &= E\{\bar{\boldsymbol{x}}^T(k)(-\alpha\boldsymbol{I})\bar{\boldsymbol{x}}(k)\} \end{aligned} \tag{41}$$

so that $E\{\bar{\boldsymbol{x}}^T(k)\boldsymbol{H}(i,m)\bar{\boldsymbol{x}}(k)\} - E\{\bar{\boldsymbol{x}}^T(k)(-\alpha\boldsymbol{I})\bar{\boldsymbol{x}}(k)\} = E\{\bar{\boldsymbol{x}}^T(k)[\boldsymbol{H}(i,m)+\alpha\boldsymbol{I}]\bar{\boldsymbol{x}}(k)\} \le 0$, and $\boldsymbol{H}(i,m) + \alpha\boldsymbol{I} \le 0.$ Then $\beta + \alpha \le 0$, where $\beta = \sup\{\beta_{\max}(i,m) = \beta_{\max}(\boldsymbol{H}(i,m)),\ i \in \mathbb{I},\ m \in \mathbb{M}\}$ is the supremum of the maximum eigenvalues of $\boldsymbol{H}(i,m)$. Since $\alpha > 0$, so that $\beta < 0$, and $\boldsymbol{H}(i,m) < 0$. Hence, the closed-loop system Eq. (30) is asymptotic mean-square stable, and $\boldsymbol{H}(i,m) < 0$.

The aforementioned asymptotic mean-square stability condition, Eq. (35) is nonlinear and difficult to be implemented in real time. A linear criterion will be introduced based on the LMIs with Schur complement.

Definition 6 [78]: Schur complements of a strict LMI $\boldsymbol{M}$ are defined as follows

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{Q}(x) & \boldsymbol{S}(x) \\ \boldsymbol{S}^T(x) & \boldsymbol{R}(x) \end{bmatrix} > 0,$$

where $\boldsymbol{Q}(x) = \boldsymbol{Q}^T(x)$, $\boldsymbol{R}(x) = \boldsymbol{R}^T(x)$, and $\boldsymbol{S}(x)$ depend affinely on x and $\boldsymbol{R}(x)$ is invertible, is equivalent to

$$\boldsymbol{Q}(x) - \boldsymbol{S}(x)\boldsymbol{R}^{-1}(x)\boldsymbol{S}^T(x) > 0, \text{ and } \boldsymbol{R}(x) > 0.$$

Theorem 3: There exists a controller that has the form as in Eqs. (17) and (18) so that the closed-loop system Eq. (34) is asymptotic mean-square stable if and only if there exists $\boldsymbol{P}(i,m) = \boldsymbol{P}^T(i,m) > 0$ satisfying

$$\begin{bmatrix} \boldsymbol{P}(i,m) & \boldsymbol{N}(i,m) \\ \boldsymbol{N}^T(i,m) & \boldsymbol{G}(j,n) \end{bmatrix} > 0 \tag{42}$$

with $\boldsymbol{N}(i,m) = \sum_{j=1}^{p}\sum_{n=1}^{q} \varsigma_{ij}^{\frac{1}{2}} \xi_{mn}^{\frac{1}{2}} (\boldsymbol{A} + \boldsymbol{BKC})^T$.

Proof: The proof is obtained by Schur complement with $\boldsymbol{G}(j,n)\boldsymbol{P}(j,n) = \boldsymbol{I}$ and Theorem 2.

The conditions in Theorem 3 are in fact a set of LMIs with non-convex constraints that can be solved by several existing algorithms with reasonable calculation efforts. However, the on-line calculation of such LMI problem with defined the coefficient matrices may require long computational time and induce more time delays to the data processing and control-law generation. Hence, an off-line calculation is adopted in this research, and experiments are conducted for its effectiveness to the NCS in the real-time sense. The control law with various levels of time delays and packet losses will be computed off-line and be tabulated for looking up during the implementation.

4.2 ALGORITHM IMPLEMENTATION

Practical NCS normally has no clock synchronization mechanism over the entire communication network. Therefore, no explicit time-delay information is available to Server and Client in real time. Similarly, no explicit packet-loss information can be

detected in real time either. All the information can be obtained by the next sampling period based on the assumption in the dissertation. Due to the stochastic nature of the communication network, the packets containing the control inputs of each loop that arrive at Client may not be in the same sequence as they were initially sent by Server. All these possibilities make it challenging to implement the controller in the practical NCSs.

By setting up a timestamp segment in the packets traveling through the communication network, the total time delays and packet losses can be detected by Client at the end of each sampling period. The total time delays can be inferred by calculating the difference between the time instance that Client sends plant-output packets to Server and the time instance that Client receives control-input packets from Server. The structure of the total time delays has the following form.

$$\Delta timestamp(k) = \tau^{ca}(k) + \tau^{sc}(k) + \tau(k), \tag{43}$$

where $\tau(k)$ includes the packet-processing time, queuing time, other calculating time, etc., on both Server and Client. Note that, in general, τ^{ca} and τ^{sc} are not necessarily the same. From Eq. (9), one can have

$$\tau^{ca}(k) \approx \tau^{sc}(k) \approx \frac{1}{2}\Delta timestamp(k) \tag{44}$$

in the controller design and implementation to be presented in Section 4.3.3. Compared to the time delays over the communication network, the packet-processing time, queuing time, or calculating time can be much smaller or neglected under certain circumstances. If Client receives no updated control signal within a certain time period, it may assume the packet has been lost.

Note that the time delays calculated by tracking the timestamps can only be accessed by the end of each sampling period, so the current time-delay information will only be able to be applied to the NCS by the next sampling period. Packet losses can be handled similarly. Server will use the time-delay information carried from the previous data packet to compensate for the effect of the time delays and packet losses a sampling period later.

An algorithm that implements the proposed output feedback controller is as follows. The implemented algorithm is illustrated in Fig. 41. The solid lines represent independent control flows on Server and Client. The dashed lines represent chronological data-packet exchanges between Server and Client. This flow chart explains the control flow in one control iteration of the NCS.

Algorithm 1: The algorithm describes the control flow in one sampling period.

(1) At the beginning of the current sampling period, Server waits for the plant-output data packet arriving from Client. The details of the data-packet structures will be given in Section 4.3.

(2) When the data packet arrives, Server first checks the corresponding data segment in the packet to verify whether a packet is lost in the previous sampling period.

(3) Server checks the time-delay states. The time-delay information is contained in the corresponding data segment. Then Server calculates the control input based on the time-delay states. Note that all the control laws for various time-delay states and packet losses are calculated off-line and tabulated on Server.

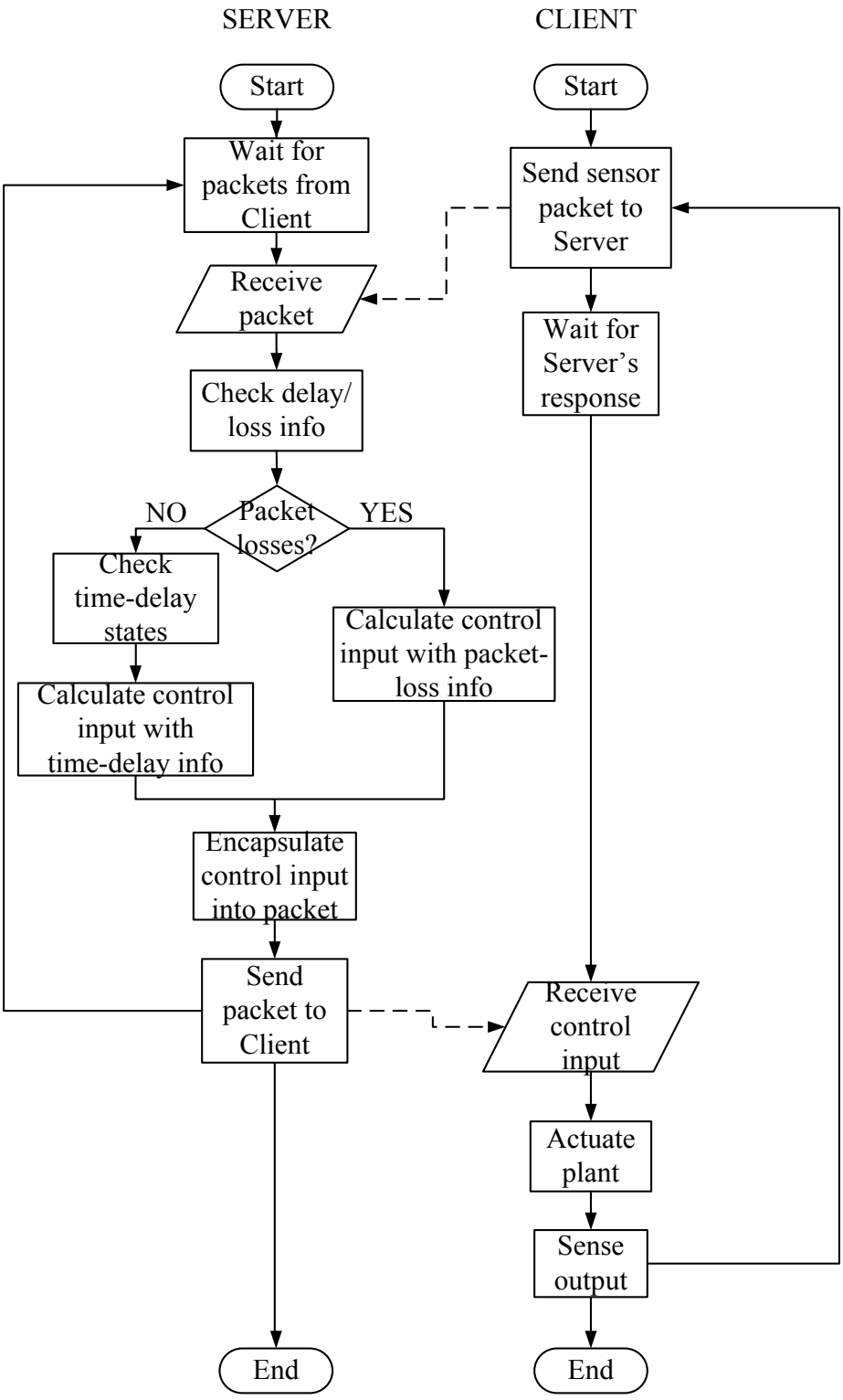


Fig. 41. Flow chart of the output feedback algorithm implementation

(4) Server sends the control-input data packet back to Client to actuate the plant. If the newly updated control-input packet is lost in the link, Client will use previous control-input data to actuate the plant.

4.3 CONTROLLER IMPLEMENTATION AND EXPERIMENTS

In this section, key experimental results are provided to verify the effectiveness of the Markov-chain-based output feedback method. The DC motor speed-control system in Fig. 13 was set up as the test bed.

4.3.1 Experimental Setup Review

Recall that Linux Redhat 7.3 with RTAI 3.4 is the OS running on Server, and Linux Ubuntu 6.10 with RTAI 3.4, on Client. Comedi is used as the drivers and libraries of data acquisition on Client. A NI PCI-6221 DAQ card enables the DC motor test bed to send out plant-output data packets and receive control-input data packets through the LAN. The speed control is achieved by controlling the output voltage of a PWM amplifier. Figure 42 shows the block diagram of the entire experimental setup.

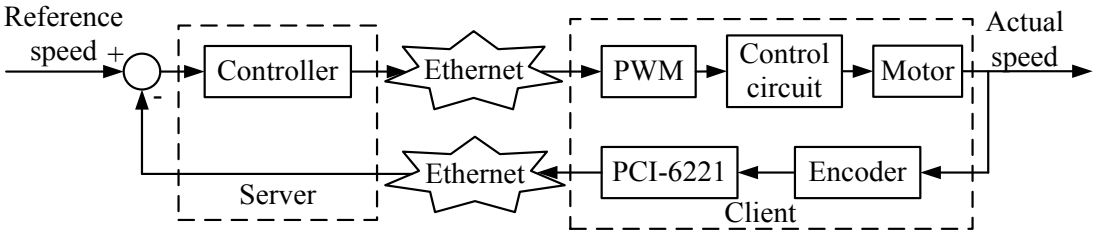


Fig. 42. Block diagram of the DC motor speed-control system

The communication network in the experiment is a 100-Mbps Ethernet with unblocked UDP sockets. The data-packet structures for both Server and Client are as follows. As shown in Fig. 43, a new segment that indicates the time-delay and packet-loss information is added to the end of the original data packet structure of Client defined in Fig. 16. The segment is used to track the random time delays and packet losses during the date packet transmission. If a packet is lost, this segment contains negative value to notify Server. If not, it contains the total time delays information.

Server	802.3 Header (14 bytes)	IP Header (20 Bytes)	UDP Header (8 Bytes)	Control Data (16 Bytes)	Timestamp (8 Bytes)	Identifier (8 Bytes)	SP (8 Bytes)		
Client	802.3 Header (14 bytes)	IP Header (20 Bytes)	UDP Header (8 Bytes)	Sensor Data (56 Bytes)	Timestamp (8 Bytes)	Identifier (8 Bytes)	BU (8 Bytes)	Type (1 Bytes)	Delay/loss info (8 Bytes)

Fig. 43. Output-feedback controller data-packet structures

4.3.2 Experimental System Modeling

Based on DC motor datasheet [79], its state-space model can be represented as

$$x_p(k+1) = -0.26x_p(k) + 2.04u(k) \tag{45}$$

$$y(k) = x_p(k), \tag{46}$$

where $u(k) \in \mathbb{R}$ is the input voltage, and $y(k) \in \mathbb{R}$ is the angular velocity, respectively.

The network-induced time delays were measured to determine the key statistical characteristics of the test bed. Recall the time-delay experiments in Fig. 24. A same random time-delay experiment was performed for 10,000 iterations. As aforementioned

in Section 4.2, the time delays attained by the experiment are the total delays in the NCS. The average of the time delays is between 0.45 and 0.5 ms, and some jitters with the average of 0.8 ms. We took these two cases as two time-delay states for the Markov-chain-based model. According to the algorithm in Section 4.2, τ^{ca} and τ^{sc} in this experiment will be one half of the total time delays as indicated in Eq. (44) so that the time-delay Markov states of τ^{ca} and τ^{sc} will be

$$\mathbb{P} = \mathbb{Q} = \{0.23, 0.4\} . \tag{47}$$

Equation (47) gives the Markov states of the experiments. The first Markov state of 0.23 ms represents the average of the time delays, and the second Markov state of 0.4 ms represents the jitters in either the controller-to-actuator link or sensor-to-controller link. τ^{ca} and τ^{sc} will take one of the values in the set. As mentioned in Section 4.2, τ^{ca} and τ^{sc} are not necessarily the same, but we assume they are since the explicit time-delay information is unavailable in the experiment. Note that the random time delays may not be exactly the same for each sampling period, and each state in the set actually represents certain time intervals. A time delay shorter than 0.35 ms represents the first Markov state, and any time delay longer than 0.35 ms, the second Markov state.

By fixing the current Markov state, the transition-probability matrix can be constructed by counting the number of the next Markov state that falls into either the first Markov state or the second Markov state in Eq. (47). The transition-probability matrices of the two Markov states are determined experimentally as

$$\boldsymbol{\Lambda} = \boldsymbol{\Gamma} = \begin{bmatrix} 0.93 & 0.07 \\ 0.75 & 0.25 \end{bmatrix}. \tag{48}$$

Equation (48) gives the probability that the time delays jump from the current Markov state to the next Markov state. (i.e., if the current time delay is 0.23 ms, then the next time delay will be 0.23 ms at 93% probability, and be 0.4 ms, 7%).

4.3.3 Controller Design and Implementation

The LMI stability criterion developed in Section 4.1 has been applied in the Matlab with the LMI Toolbox, and the V-K iteration algorithm in [34] with the following initial $\boldsymbol{P}$ matrix. The matrix $\boldsymbol{P}(i,m)$ depends on the Markov states of τ^{ca} and τ^{sc}. For instance, if τ^{ca} is at the first Markov state of 0.23 ms, and τ^{sc} is at the second Markov state of 0.4 ms, $\boldsymbol{P}(i,m)$ will be denoted as $\boldsymbol{P}(1,2)$. Set a state vector $\boldsymbol{w}_{12}=[0.23 \quad 0.4]^T$ for $\boldsymbol{P}(1,2)$, and define $\boldsymbol{P}(1,2) \triangleq \gamma \cdot diag(\boldsymbol{w}_{12}\boldsymbol{w}_{12}^T) \otimes \boldsymbol{I}$, where γ is a weight coefficient for the optimization and $\otimes$ is the Kronecker product. The dimension of $\boldsymbol{I}$ depends on the problems, where $\boldsymbol{I}$ is 4×4 identity matrix in our experiments. All the other $\boldsymbol{P}(i,m)$ can be constructed in the same way. These initial $\boldsymbol{P}(i,m)$ will be applied to start the LMI solver and V-K iteration algorithm, which will converge to the final states at the end of all the iterations or when the errors satisfy a pre-set error bound. The choice of initial $\boldsymbol{P}(i,m)$ may vary. The convergence of the V-K iteration algorithm can be referred to [34]. Then with solving Eq. (42) using the Matlab LMI Toolbox and V-K iteration algorithm with the corresponding constraints, the controller can be designed.

The controllers are designed as presented in Table 7. The 4-tuple $\{\tau^{ca}, \tau^{sc}, \delta^{ca}, \delta^{sc}\}$ in Table 7 represents different Markov states of the random time delays and packet losses as defined in Section 4.1. The order of the controller can be set

as needed. A higher-order controller may promise more robust system performance but require more computational efforts and bring more complexity to the system. In our experiments, the plant represented with Eqs. (45) and (46) is a first-order system. We design the controller to be first-order, so the whole closed-loop system is second-order.

Table 7. Output feedback controller parameters

$\{\tau^{ca}, \tau^{sc}, \delta^{ca}, \delta^{sc}\}$	A_c	B_c	C_c	D_c
{0.23, 0.23, 1, 1}	1.0102	0.9687	0.0396	1.7621
{0.23, 0.4, 1, 1}	1.0155	0.9879	0.0408	1.7889
{0.4, 0.23, 1, 1}	1.0155	0.9879	0.0408	1.7889
{0.4, 0.4, 1, 1}	1.0412	1.0030	0.0421	1.8162
{–, –, 0, 0}	1.1974	1.1534	0.0557	2.0886

As mentioned in Section 4.2, when the packet is lost, no time-delay information will be available. The 4-tuple {–, –, 0, 0} represents the case that both the controller-to-actuator and the sensor-to-controller packets are lost. The sensor-to-controller packet loss is represented by {–, –, 1, 0}. However, Server will not be able to calculate the updated control input since it has not received any newly updated output information. The other case of the controller-to-actuator packet loss is represented by {–, –, 0, 1}. When this happens, the updated control input is calculated by Server but cannot arrive at Client. Therefore, all these cases can be grouped into the case {–, –, 0, 0} in the experiments since Client will not receive any updated control input for these three cases.

For the three consecutive packets losses case, two second-order AR models are applied to predict the lost plant outputs and control inputs so that

$$\hat{\boldsymbol{y}}(k) = 0.6621\boldsymbol{y}(k-1) + 0.3377\boldsymbol{y}(k-2), \tag{49}$$

$$\hat{\boldsymbol{y}}(k+1) = 0.6867\boldsymbol{y}(k-1) + 0.3132\boldsymbol{y}(k-2), \tag{50}$$

$$\hat{\boldsymbol{y}}(k+2) = 0.7151\boldsymbol{y}(k-1) + 0.2848\boldsymbol{y}(k-2), \tag{51}$$

$$\hat{\boldsymbol{u}}(k) = 0.5094\boldsymbol{u}(k-1) + 0.4094\boldsymbol{u}(k-2), \tag{52}$$

$$\hat{\boldsymbol{u}}(k+1) = 0.6136\boldsymbol{u}(k-1) + 0.3862\boldsymbol{u}(k-2), \tag{53}$$

$$\hat{\boldsymbol{u}}(k+2) = 0.6849\boldsymbol{u}(k-1) + 0.3151\boldsymbol{u}(k-2). \tag{54}$$

As discussed earlier, the order of the AR model depends on the number of the states of the Markov chain. Here, in the experiments, the time-delay Markov chain has two independent states as in Eq. (47). Therefore, the order of the AR model will be two in the experiments. The AR models are calculated in Matlab. The best-fit values of the above AR models are 83.3220 and 83.6391, respectively.

4.3.4 Experimental Results

The system performance with the proposed method is used to compare the performance with that of the PI controller in [64]. The difference equation of the PI controller is defined in Eq. (14).

All experiments were executed with a 3-ms sampling period for 500 iterations. The reference speed of the DC motor in all the experiments was set to be 10 rps. Three separate experiments, without packet losses, with 10% single packet losses, and with 20% consecutive packet losses, were conducted to evaluate the effectiveness of the

proposed method. All the experiments were executed under the same network condition as the time-delay experiment as measured in Eq. (47). The Ethernet LAN in the lab was robust so that no packet losses occurred even with UDP. Therefore artificial packet losses were introduced to the NCS with an approximate 10% and 20% loss rate, respectively. For instance, for 10% loss rate, a random function that takes value from 0 to 1 was introduced, and a threshold of 0.1 (10% loss rate) was set for the comparison. If the random number was less than the threshold, the packet would be dropped from the NCS. Note that the random modulo operation does not generate a truly uniformly distributed random number in [0, 1], but it is generally a good approximation. Since we run the experiments with a large number of iterations, we assume that the packet-loss rate is about 10%. Similarly, for the consecutive-packet-loss case, a 20% three consecutive packets losses were artificially introduced in to the NCS. To clearly see the effects of the 20% three consecutive packets losses, the packet losses were introduced to the NCS at 500 ms.

The step responses of the NCS are shown in Figs. 44–46. Figure 44 shows the results of the PI controller and the proposed controller without artificial packet losses. Without packet losses, the steady-state errors of the conventional PI control and the method proposed in this section are almost the same. Figure 45 shows the experimental results with 10% random artificial packet losses. Figure 46 shows the experimental results with 20% three consecutive packets losses. As shown in Figs. 45 and 46, even when packets were lost in the communication network, our approach could track the reference command faithfully whereas the PI controller could not compensate for the

random time delays and packet losses. The proposed method not only uses predictive control data but also compensates for the effect of packet losses. Hence the system performance can be enhanced.

Figures 47–49 show the DIAE of all the experiments with the proposed method and the PI controller. Each figure shows the DIAE of the experimental data without packet losses, with 10% single packet losses, and 20% consecutive three-packet losses, respectively. From these figures, one can see that system errors of the NCS dramatically increase when packets are lost in the communication network. From Figs. 47–49, the proposed method reduced the DIAE by about 13% without packet losses. For the single packet losses case, the proposed method reduced the DIAE by as much as 30% compared to the PI controller. The three consecutive packets losses case, the proposed method could still reduce the DIAE of the NCS by about 40%. In all these results, the Markov-chain-based method proposed in this research exhibited satisfactory system performance.

4.4 SUMMARY

This section proposed an output feedback method for the stabilization and control of the NCS with random time delays and packet losses. By modeling the random time delays with time-homogeneous Markov chains and packet losses with Dirac delta functions, the closed-loop system was stabilized, and the performance was much enhanced compared to a conventional control method. An asymptotic mean-square stability criterion for the NCS was obtained in terms of a Lyapunov function and a set of LMIs with matrix constraints. An algorithm implementation of the stability criterion was

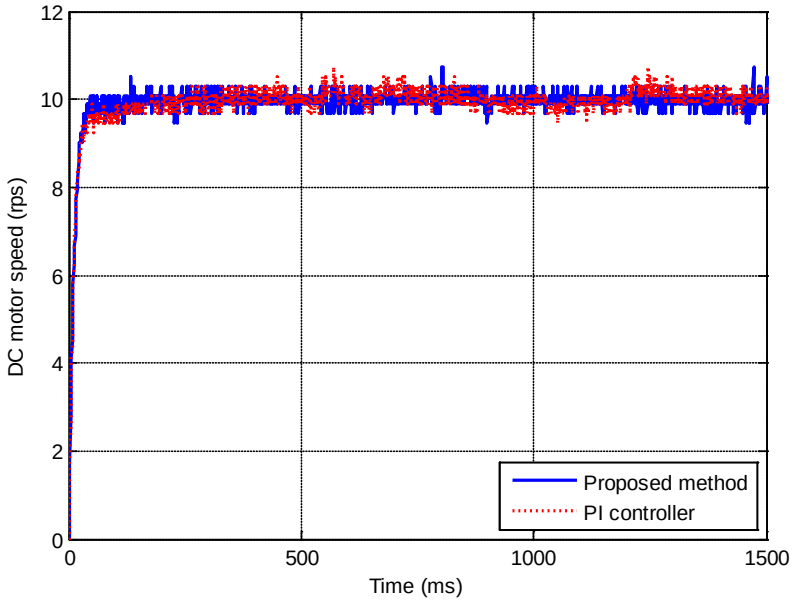


Fig. 44. Step responses of Client 2 without packet losses

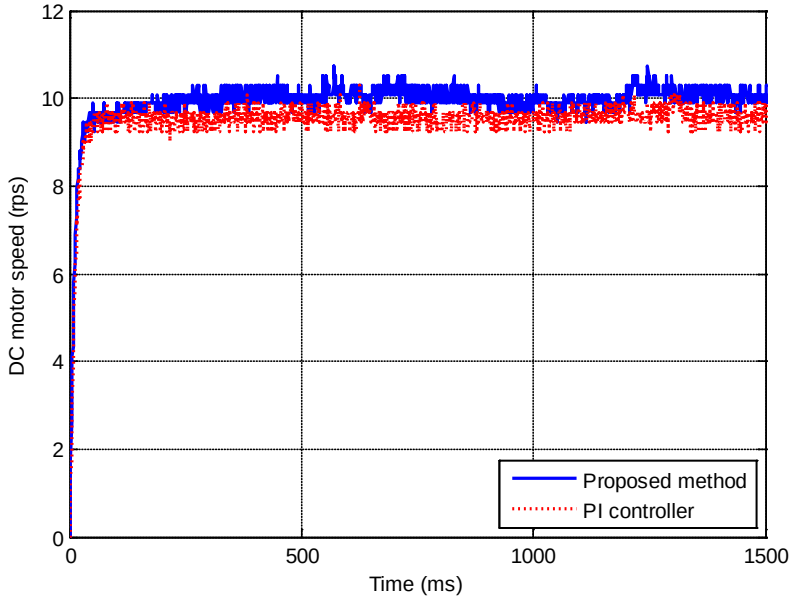


Fig. 45. Step responses of Client 2 with 10% single packet losses

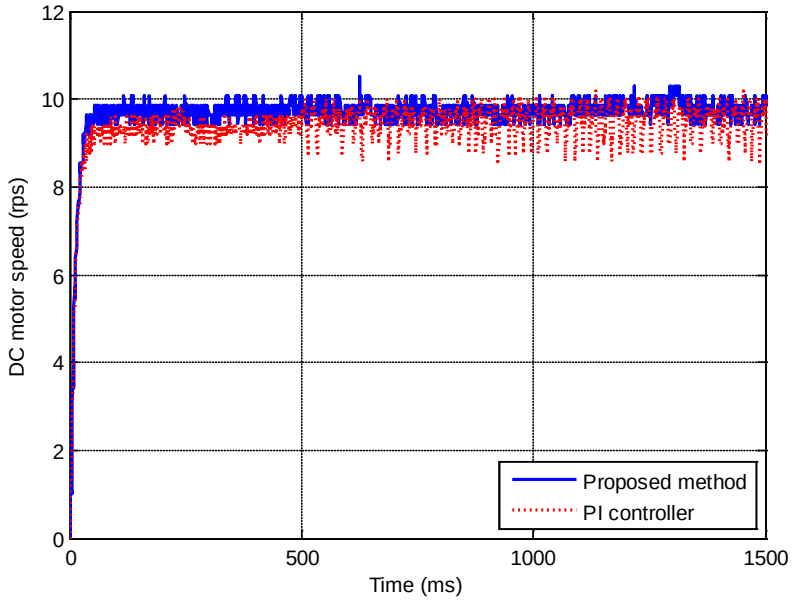


Fig. 46. Step responses of Client 2 with 20% three consecutive packets losses

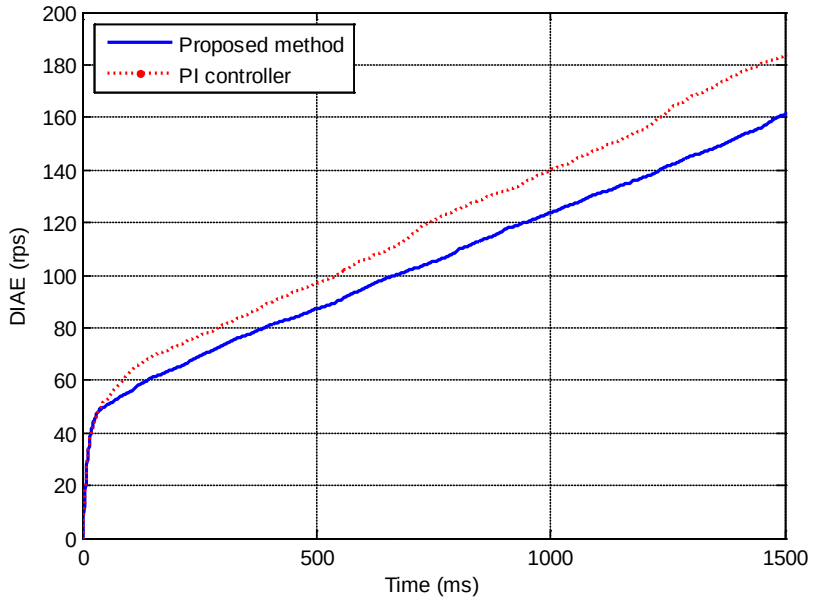


Fig. 47. DIAE of the proposed method vs. PI controller without packet losses

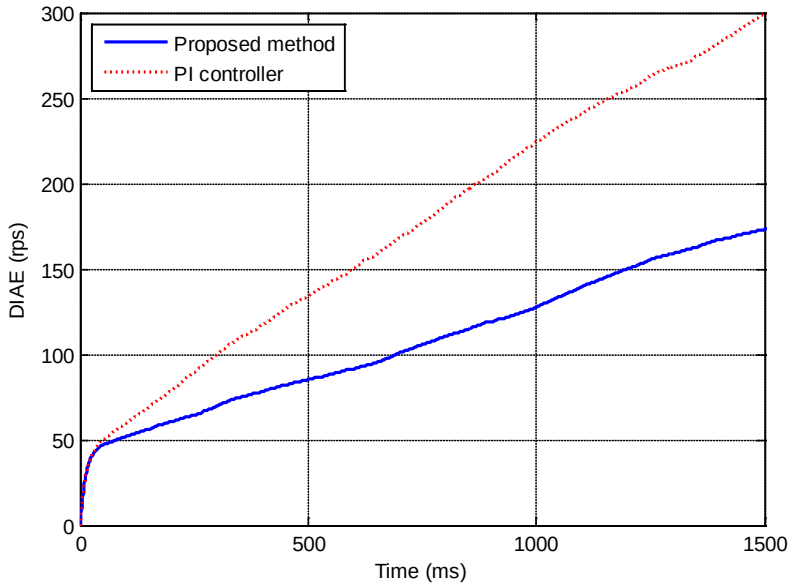


Fig. 48. DIAE of the proposed method vs. PI controller with 10% single packet losses

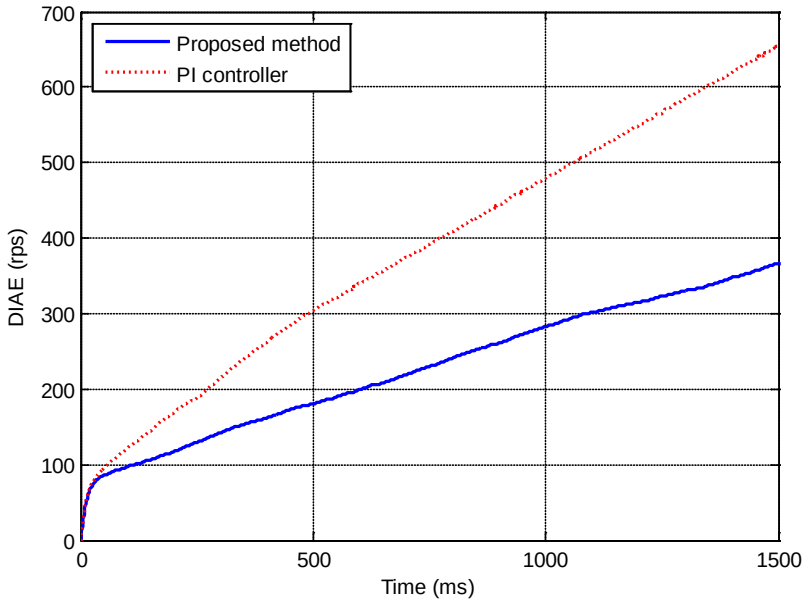


Fig. 49. DIAE of the proposed method vs. PI controller with 20% three consecutive packets losses

also presented in this section. The experimental results demonstrated the feasibility and effectiveness of the proposed method. The proposed method enhanced the system performance with and without packet losses compared to a conventional control algorithm. The DIAE without packet losses was reduced by 13% with the proposed method. The DIAE with 10% single packet losses was reduced by about 30%, and the DIAE with 20% three consecutive packets losses, by about 40%. The NCS could track the reference command faithfully with the proposed method when random time delays and packet losses existed in the links whereas the NCS failed to track the reference command with a conventional control algorithm.

5. OPTIMAL BANDWIDTH ALLOCATION AND SCHEDULING OF THE NCS

Traditionally, a controller design problem is separated from software design and implementation. This separation allows control and computer communities to focus on their specific problems. Controller designers disregard the characteristics of the computational and communication resources but mainly focus on the controller and system stability and performance. On the other hand, real-time OS designers consider the control loops as periodic tasks with hard deadlines. They focus more on how to schedule all the tasks and guarantee that the tasks do not miss the deadlines [37]. In the NCS, however, these two fields are correlated in a closer way so that their separation will lead to poor system performances. The ideal linear relation between the system performance and the sampling frequency is no longer the case for the NCS design because of the existence of the network. A larger sampling frequency will increase the number of data packets in the network, which will bring longer time delays and might even overload and destabilize the network.

In general, multiple controllers or multiple clients can coexist in the same NCS. A representative framework of an NCS was shown in Fig. 3. In this framework, the NCS includes several operation scenarios—a single controller controls a single client, a single controller controls multiple clients, and multiple controllers collaboratively control a single client. All the clients will compete for the limited resources in the NCS to maintain the stability and performance. Therefore, the communicational and computational resource allocation and scheduling plays a crucial role in an NCS. Guan

et al. considered additive colored white Gaussian noise when optimizing the performance of the NCS with limited bandwidth [80]. A dynamic bandwidth allocation algorithm based on captured visual content information was presented to raise the bandwidth utilization of an NCS [81]. Heemels et al. presented a general framework that incorporated communication constraints, varying transmission intervals and varying delays. Based on a newly developed NCS model including all these above network phenomena, the authors provided an explicit construction of a continuum of Lyapunov functions [82]. Xu et al. formulated a bandwidth optimization and scheduling algorithm of the NCS based on a non-cooperative game model in [83]. The existence and uniqueness of Nash equilibrium point are proved.

Traditionally, digital control assumes that the system performance can be reflected by a monotonically decreasing linear or exponential function of the sampling frequency. However, this is no longer the case for the design of an NCS as discussed in Section 1. A higher sampling frequency will increase the number of data packets in the network, which will cause longer time delays and might even overload and destabilize the network. Therefore, the linear models of the system performance proposed in the aforementioned literatures could not completely represent the system dynamics in an NCS. The effects on the system performance from the possible longer time delays brought by a high sampling frequency should be considered when formulating the performance index function (PIF) of an NCS. To better discuss the system performance and achieve the bandwidth allocation of an NCS, approximations of the PIF which can fully reveal the characteristics of an NCS are necessary. These approximations should

include the effects of the time delays brought by high sampling frequencies as an essential part when setting the system PIF of an NCS. Not only the time delays but also the scheduling sequences of controlled plants can affect the PIF of an NCS. Hence, two system approximations, exponential and quadratic, which consider the effects of time delays as a crucial part of the system PIF, are proposed to achieve the optimal bandwidth allocation and scheduling of an NCS. The proposed approximations and the optimal solutions are expected to exhaust the entire network bandwidth available to the NCS to maximize the BU and the system performance. Note that although the proposed approximations and scheduling algorithms are mainly for an NCS with SSMC framework, they can be applied to an NCS with MSMC framework easily with proper adjustments.

5.1. SYSTEM PERFORMANCE APPROXIMATIONS

Consider an NCS of a framework that contains one server and multiple clients. To guarantee the stability and enhance the system performance, all the clients are assumed to compete for the CPU time and the network bandwidth to calculate control inputs and transmit data packets. Accordingly, the most common objective in the resource allocation of an NCS is to optimize the overall quality of control subjected to certain resource limitations.

Recall Fig. 3 in Section 1, the system performance of an NCS is no longer a linear function of the sampling frequency. As the sampling frequency increases, the system performance will be degraded due to large amount of data packets transmitted in the network. To be consistent with simulation and experiments in this section, Fig. 50

gives an intuitive idea about trends of the system performance of an NCS regarding to the sampling frequency. Note that Fig. 50 reflects the generic illustration of Fig. 3. In Fig. 50, f_γ is the optimal sampling frequency that yields the optimal system performance of an NCS. f_α and f_β are the sampling frequency boundaries of the acceptable performance range. The acceptable performance depends on the users' requirements. It is not necessary the stable region of the NCS. Therefore, f_α and f_β may not necessary be the minimum and maximum sampling frequencies of the NCS, respectively.

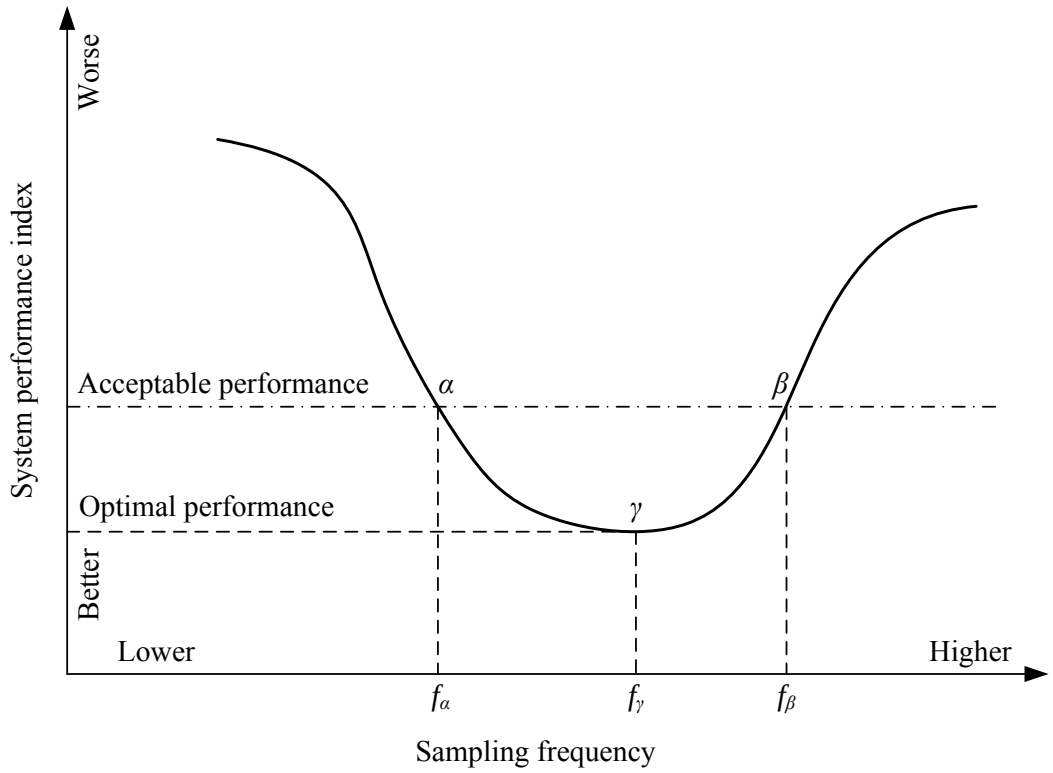


Fig. 50. NCS performance index vs. sampling frequency

5.1.1. Network Bandwidth of the NCS

To achieve the optimal resource-allocation objective, a system PIF in terms of various resources is set up. Based on the definition of the network bandwidth in [26], the relation between the sampling frequency and the BU can be indicated by the following equation,

$$b_i^k = \tau_i^k f_i^k, \tag{55}$$

where b_i^k is the BU, f_i^k is the sampling frequency, and τ_i^k is the total time delay in the NCS that includes the propagation delay from the network and the data processing time. The subscript i indicates the index of the clients in the NCS, and the superscript k indicates the control iterations.

Note that given a certain amount of time delay, Eq. (55) gives a means to evaluate the clients' sampling frequencies and represents the portion of network bandwidth assigned to each plant. Since τ_i^k includes the data processing time on Server's CPU, this bandwidth definition also implicitly indicates the CPU resource allocation on Server. In control system design, the sampling frequency directly relates to the system stability and performance. Equation (55) also gives an implicit means to measure a client's stability and performance. A large BU implies a high sampling frequency so that a client will have a better performance. However, an upper bound exists on the NCS bandwidth. If the BU reaches the network bandwidth saturation threshold, the network will be overloaded and induce more time delays or packet losses, and the performance of an NCS will be degraded.

5.1.2. Performance Index Function

Recall that the DIAE is adopted to be the performance index formulated in Section 3. For each individual plant, at various sampling frequency, the DIAE will take a different value. Hence, a set of accumulated DIAEs of a client over a stability range of sampling frequencies will imply the performance of an NCS and can be applied to find the optimal sampling frequency of the client. Hereafter, the practical PIF will be defined as a piecewise function of the sampling frequency as follows,

$$\overline{J}_i(f_i) = \sum_{k=k_0}^{k_f} \left| e_i^k(f_i) \right|. \tag{56}$$

Two approximations will be proposed to capture the trends of the practical PIF as in Eq. (56) so that the analytical optimal bandwidth allocation can be achieved.

5.1.3. Exponential Approximation Modeling

The NCS system performance without considering time delays can be approximated as an exponential function [84–85]. However, the negligence of the time delay in the approximation would not reveal the characteristics of the NCS. Hence, a PIF considering the effects of the time delays is necessary and can be defined as

$$J_i(f_i^k, \tau_i^k) = d \exp(a f_i^{\,k} + b\tau_i^k + c) + g\,, \tag{57}$$

where a, b, c, d, and g are approximation coefficients. The time delays depend on many aspects such as the data-packet size, number of packets in the network, network conditions, router's capacity, unpredictable uncertainties, etc. Although the time delays affect the system performance, they are not directly controllable variables in a design of an NCS. However, by controlling the sampling frequency of each client in an NCS, the

number of data packets in the network can be maintained at a certain level so that the average of time delays can be controlled within a certain range. Therefore, for simplicity, we assume that the effects of the time delays can be reflected by an increasing function of the sampling frequencies of the plants in an NCS. The details of system performance versus the time delays can refer to [59, 86–87]. Hence, from an NCS design perspective, the PIF that reveals the effects of the time delays brought by a high sampling frequency can be revised as an increasing exponential function of the sampling frequency. Hereafter, from a traditional digital design perspective, $E_i(f_i^k, t)$ defines an approximated PIF of Client i as a decreasing exponential function of the sampling frequency. Similarly, from an NCS design perspective, $F_i(f_i^k, t)$ defines an approximated PIF of Plant i as an increasing exponential function of the sampling frequency. Therefore,

$$E_i(f_i^k, t) = e^{-\beta_i f_i^k}, \tag{58}$$

and

$$F_i(f_i^k, t) = e^{\delta_i f_i^k}, \tag{59}$$

where β_i and δ_i are the approximation coefficients. These parameters can be obtained from simulation or experiments by a LSM approach. Refer to Eqs. (101) and (103) in Section 5.3 as examples. Therefore, for each individual client, the PIF can be defined as

$$\overline{J}_i \cong J_i = \sum_{k=0}^{M-1} (\alpha_i E_i(f_i^k, t) + \gamma_i F_i(f_i^k, t)) dt, \tag{60}$$

where M is the maximum control iteration. The coefficients α_i and γ_i balance the impacts of the errors and time delays in the PIF of the corresponding client.

For the NCS, the purpose of optimal bandwidth allocation is to minimize the PIF

$$\begin{aligned}\min_{f\in\Omega} J &= \min_{f\in\Omega} \sum_{i=1}^{N} \omega_i^k J_i \\ &= \min_{f\in\Omega} \sum_{i=1}^{N}\sum_{k=0}^{M-1} \omega_i^k (\alpha_i E_i(b_i^k,\ t) + \gamma_i F_i(b_i^k,\ t))dt \\ &= \min_{f\in\Omega} \sum_{i=1}^{N}\sum_{k=0}^{M-1} \omega_i^k (\alpha_i e^{-\beta_i f_i{}^k} + \gamma_i e^{\delta_i f_i{}^k})dt, \end{aligned} \tag{61}$$

$$\text{subject to } \sum_{i=1}^{N} \tau_i^k f_i{}^k \le B, \qquad \forall k = 1, \cdots, M \tag{62}$$

where $0 \le B \le 1$ is the network bandwidth saturation threshold in the NCS, N is the number of clients, ω_i^k is the weight for Client i at control iteration k, and Ω is the set of sampling frequencies that maintains the stability of the clients. The selection of ω_i^k can be based on the system requirements. For example, the client with the largest sampling frequency may indicate the difficulties in maintaining the stability and system performance and wins the largest weight. Furthermore, the PIF is a convex function of sampling frequencies, and it is this convexity that allows for the optimal sampling frequency for a set of clients with appropriate convex optimization methodologies.

5.1.4. Quadratic Approximation Modeling

In Section 5.3, one can see that the exponential approximation can closely approximate the practical system performance, but a closed-form optimal solution of Eqs. (61–62) is not easy to obtain in real time. Therefore, a quadratic approximation is

proposed as a replacement of the exponential approximation. The quadratic approximation has a simple closed-form optimal solution to Eqs. (64–65) in the below. For each individual plant, the PIF can be defined as

$$\overline{J}_i \cong J_i = \sum_{k=0}^{M-1} (a_i (f_i^k)^2 + b_i f_i^k + c_i) dt , \tag{63}$$

where a_i, b_i, and c_i are the approximation coefficients. Refer to Eqs. (102) and (104) in Section 5.3 as examples.

For the entire NCS, the objective function and constraints could be formulated as

$$\begin{aligned} \min_{f \in \Omega} J &= \min_{f \in \Omega} \sum_{i=1}^{N} \omega_i^k J_i \\ &= \min_{f \in \Omega} \sum_{i=1}^{N} \sum_{k=0}^{M-1} \omega_i^k (a_i (f_i^k)^2 + b_i f_i^k + c_i) dt , \end{aligned} \tag{64}$$

$$\text{subject to } \sum_{i=1}^{N} \tau_i^k f_i^k \le B , \qquad \forall k = 1, \cdots, M \tag{65}$$

Note that this quadratic PIF is also a convex function of the sampling frequency.

5.2. OPTIMAL BANDWIDTH ALLOCATION AND SCHEDULING

In this section, the optimal solution of the proposed exponential and quadratic approximations and the scheduling of the bandwidth assignment sequence of the clients are given. To facilitate the development, the following assumptions are made.

Assumption 1: The total time delay τ_i^k in Eq. (55) is a random variable by the nature of the network. For the simplicity of analysis and optimization, however, it is assumed to be a constant at each different sampling frequency f_i of Client *i*, and an

average value is used. Then the superscript k in all the approximations can be dropped, and $\overline{\tau}_i$ is the average time delay for Client i.

Assumption 2: All the clients can be scheduled at their minimum sampling frequency. That is, when $f_i = f_i^{\min}$, we have $\sum_{i=1}^{N} \overline{\tau}_i f_i^{\min} \le B$, where $f_i^{\min}$ is the minimum sampling frequency of the Client i. When all the clients are at their maximum BU or maximum sampling frequency, the total BU of the entire NCS may or may not exceed the network bandwidth saturation threshold B.

An NCS could contain various clients that have different system specifications and requirements. These clients can be categorized into two groups, the one with variant sampling frequencies, and the one with fixed sampling frequencies. If an NCS includes both groups of clients, the bandwidth threshold B needs to be modified as $\hat{B} = B - \sum_{j \in \mathbb{J}} \overline{\tau}_j f_j$, where $\mathbb{J}$ is the set of the indices of the clients with fixed sampling frequencies. Then, for the rest of the clients with variant sampling frequencies, the new bandwidth threshold $\hat{B}$ will be used for the optimization purpose so that the objective function of Eq. (61) or (64) can still be applied. Or if a certain percentage of the network bandwidth should be reserved for other functionalities, the newly defined $\hat{B}$ can also be applied so that $\hat{B} = B - \tilde{B}$, where $\tilde{B}$ is a reserved network bandwidth.

5.2.1. Optimal Solution of Exponential Approximation

The two approximations discussed in the previous sections are both convex functions. Note that the constraints of the objective PIF approximations are also convex

functions. Hence, the convex optimization techniques can be applied to solve for optimal solutions of each approximation. Note that the exponential and quadratic approximations proposed in this section are nonlinear functions. Therefore, the KKT condition will be applied to solve the optimization approximations in Eqs. (61–62) and (64–65).

Theorem 4 [88]: Let $\boldsymbol{x}^* \in \mathbb{R}^n$ be a minimum solution of the problem

$$\text{minimize } f(\boldsymbol{x})$$

$$\text{subject to } \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}$$

$$\boldsymbol{g}(\boldsymbol{x}) \le \boldsymbol{0},$$

and suppose $\boldsymbol{x} \in \mathbb{R}^n$ is a regular point for the constraints. Then there is a vector $\boldsymbol{\lambda} \in \mathbb{R}^m$ and a vector $\boldsymbol{\mu} \in \mathbb{R}^p$ with $\boldsymbol{\mu} \ge \boldsymbol{0}$ such that

$$\nabla f(\boldsymbol{x}^*) + \boldsymbol{\lambda}^T \nabla \boldsymbol{h}(\boldsymbol{x}^*) + \boldsymbol{\mu}^T \nabla \boldsymbol{g}(\boldsymbol{x}^*) = \boldsymbol{0},$$

$$\boldsymbol{\mu}^T \boldsymbol{g}(\boldsymbol{x}^*) = \boldsymbol{0}.$$

For the exponential approximation defined in Eqs. (60–61), the optimal solution will be given by the following theorem.

Theorem 5: Given an NCS with *N* clients, and with the PIF approximated in Eqs. (61–62), an optimal solution, is given by

$$f_i^* = f_i^{\min}, \qquad i = 1, \cdots, l \tag{66}$$

$$f_j^* = g_j(\lambda), \qquad j = l+1, \cdots, N \tag{67}$$

where *l* is the smallest index so that

$$\sum_{i=1}^{l} \overline{\tau}_i f_i^{\min} + \sum_{j=l+1}^{N} \overline{\tau}_j f_j^{\,*} \ge B, \tag{68}$$

and $g_j(\lambda)$ is the solution to

$$\Xi_i e^{-\beta_i f_i} + \Phi_i e^{\delta_i f_i} + \lambda \overline{\tau}_i = 0, \tag{69}$$

where $\Xi_i = -\omega_i \alpha_i \beta_i$ and $\Phi_i = \omega_i \gamma_i \delta_i$.

Proof: The KKT condition and the Lagrange multipliers λ, λ_{i1}, and λ_{i2} will be introduced. Then define the Lagrange equation as

$$L = \sum_{i=1}^{N} \omega_i (\alpha_i e^{-\beta_i f_i} + \gamma_i e^{\delta_i f_i}) + \lambda (\sum_{i=1}^{N} \overline{\tau}_i f_i - B) + \sum_{i=1}^{N} \lambda_{i1} (f_i^{\min} - f_i) + \sum_{i=1}^{N} \lambda_{i2} (f_i - f_i^{\max}) \tag{70}$$

Then from the KKT condition, the dual feasibility is

$$\omega_i (-\alpha_i \beta_i e^{-\beta_i f_i} + \gamma_i \delta_i e^{\delta_i f_i}) + \lambda \overline{\tau}_i - \lambda_{i1} + \lambda_{i2} = 0, \tag{71}$$

and the complementary slackness are

$$\lambda (\sum_{i=1}^{N} \overline{\tau}_i f_i - B) = 0 \tag{72}$$

$$\lambda_{i1} (f_i^{\min} - f_i) = 0 \tag{73}$$

$$\lambda_{i2} (f_i - f_i^{\max}) = 0, \tag{74}$$

$$\lambda \geq 0 \tag{75}$$

$$\lambda_{i1} \geq 0 \tag{76}$$

$$\lambda_{i2} \geq 0, \tag{77}$$

where $i = 1, 2, \cdots, N$.

Based on *Assumption 2*, $\sum_{i=1}^{N} \overline{\tau}_i f_i^{\min} \leq B$, all the clients are given initially the minimum frequencies, $f_i = f_i^{\min}$, and there will be idle network bandwidth available. If

$\sum_{i=1}^{N} \overline{\tau}_i f_i^{\min} = B$, then the optimal solution of the objective and exponential approximation Eqs. (61–62) is $f_i^* = f_i^{\min}$. If $\sum_{i=1}^{N} \overline{\tau}_i f_i^{\min} < B$, then some or all sampling frequencies of the clients must be increased from their minimum values. For those clients that have increased sampling frequencies, their constraints are inactive so that $\lambda_{i1} = 0$ and $\lambda_{i2} = 0$ based on the KKT conditions. Therefore from Eq. (71), we have

$$\Xi_i e^{-\beta_i f_i} + \Phi_i e^{\delta_i f_i} + \lambda \overline{\tau}_i = 0, \quad i = l+1, \cdots, N \tag{78}$$

where $\Xi_i = -\omega_i \alpha_i \beta_i$ and $\Phi_i = \omega_i \gamma_i \delta_i$. Solve f_i from Eq. (78) in terms of λ, assume that λ is given at the moment and is a constant during the calculation. Multiple $e^{-\beta_i f_i}$ on both sides of Eq. (78), and let $u = e^{-f_i}$, then we will have

$$\Phi_i u^{\delta_i + \beta_i} + \lambda \overline{\tau}_i u^{\beta_i} + \Xi_i = 0, \quad i = l+1, \cdots, N. \tag{79}$$

Note that Eq. (79) is a higher-order polynomial function. If the order of Eq. (79) is higher than 5, there is no closed-form solution. The Newton-Raphson Method [84] can be applied to find roots of Eq. (79).

Let u^* be the solution of Eq. (78) and define $f_i^* = \ln(u^*)$, then

$$\sum_{i=1}^{l} \overline{\tau}_i f_i^{\min} + \sum_{j=l+1}^{N} \overline{\tau}_j \ln(u^*) \geq B. \tag{80}$$

And solve for λ from Eq. (79) with u^* solved from Eq. (80), which yields the optimal solution of the exponential approximation.

Note that Eq. (78) is a transcendental equation, and a closed-form solution for f_i may not be easily obtained. Compared to solving the transcendental equation Eq. (78), solving of the polynomial function Eq. (79) requires less computational effort.

5.2.2. Optimal Solution of Quadratic Approximation

For the quadratic approximation defined in Eqs. (64–65), the optimal solution will be given as follows.

Theorem 6: Given an NCS with *N* clients, and with the PIF in Eqs. (64–65), an optimal solution, is given by

$$f_i^* = f_i^{\min}, \qquad i = 1, \cdots, l \tag{81}$$

$$f_j^* = \frac{-\lambda \bar{\tau}_j - \omega_j b_j}{2\omega_j \alpha_j}, \qquad j = l+1, \cdots, N \tag{82}$$

where *l* is the same as in *Theorem 4* and

$$\lambda = \frac{\sum_{i=1}^{l} \bar{\tau}_i f_i^{\min} - \sum_{j=l+1}^{N} \frac{\bar{\tau}_j b_j}{2a_j} - B}{\sum_{j=l+1}^{N} \frac{\bar{\tau}_j^2}{2\omega_j a_j}}. \tag{83}$$

Proof: The KKT condition and the Lagrange multipliers λ, λ_{i1}, and λ_{i2} will be introduced. Then define the Lagrange equation as

$$L = \sum_{i=1}^{N} \omega_i (a_i f_i^2 + b_i f_i + c_i) + \lambda (\sum_{i=1}^{N} \bar{\tau}_i f_i - B) + \sum_{i=1}^{N} \lambda_{i1} (f_i^{\min} - f_i) + \sum_{i=1}^{N} \lambda_{i2} (f_i - f_i^{\max}) \tag{84}$$

From the KKT condition, the dual feasibility of Eq. (84) is

$$\omega_i (2a_i f_i + b_i) + \lambda \bar{\tau}_i - \lambda_{i1} f_i + \lambda_{i2} f_i = 0, \tag{85}$$

and the complementary slackness are

$$\lambda(\sum_{i=1}^{N}\overline{\tau}_i f_i - B) = 0 \tag{86}$$

$$\lambda_{i1}(f_i^{\min} - f_i) = 0 \tag{87}$$

$$\lambda_{i2}(f_i - f_i^{\max}) = 0, \tag{88}$$

$$\lambda \geq 0 \tag{89}$$

$$\lambda_{i1} \geq 0 \tag{90}$$

$$\lambda_{i2} \geq 0, \tag{91}$$

where $i = 1,\ 2,\ \cdots,\ N$.

Similarly, based on *Assumption 2*, all the clients are given initially the minimum frequencies, $f_i = f_i^{\min}$, and there will be idle network bandwidth available. Similar, for those clients, which have sampling frequencies other than their minimum values, $\lambda_{i1} = 0$ and $\lambda_{i2} = 0$ based on the KKT conditions. Therefore from Eq. (85), we have

$$2\omega_i a_i f_i + \omega_i b_i + \lambda\overline{\tau}_i = 0, \quad i = l+1,\ \cdots,\ N \tag{92}$$

so that

$$f_i^* = \frac{-\lambda\overline{\tau}_i - \omega_i b_i}{2\omega_i a_i}, \qquad i = l+1,\ \cdots,\ N \tag{93}$$

Substitute Eq. (93) into Eq. (86), and solve for λ.

$$\sum_{i=1}^{l}\overline{\tau}_i f_i^{\min} + \sum_{j=l+1}^{N}\overline{\tau}_j \frac{-\lambda\overline{\tau}_j - \omega_j b_j}{2\omega_j a_j} = B, \tag{94}$$

and

$$\lambda = \frac{\sum_{i=1}^{l} \overline{\tau}_i f_i^{\min} - \sum_{j=l+1}^{N} \frac{\overline{\tau}_j b_j}{2a_j} - B}{\sum_{j=l+1}^{N} \frac{\overline{\tau}_j^2}{2\omega_j a_j}}. \tag{95}$$

Note that the solutions to Eqs. (61–62) and (64–65) may vary depending on the selection of the weights ω_i^k and the approximation coefficients. Finding optimal solutions with the chosen weights and approximation coefficients may not be feasible. Then new weights and approximation coefficients need to be chosen to fulfill the feasibility of the optimization.

5.2.3. Unique Global Optimal Solution

Note that the exponential and quadratic functions are convex functions. The additional operation preserves the convexity of functions. Hence, the two proposed approximations defined in Eqs. (61–62) and (64–65) are convex approximation. With the convexity of the proposed approximations, the following theorem exists.

Theorem 7: Given the two approximations in Eqs. (61–62) and (64–65), the optimal solutions in Eqs. (66–67) and (81–82) will be the unique global optimal solutions if the solutions exist.

Proof: Consider a convex optimization problem as follows

$$\text{minimize } f(\boldsymbol{x})$$

$$\text{subject to } \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}$$

$$\boldsymbol{g}(\boldsymbol{x}) \le \boldsymbol{0},$$

and let $\boldsymbol{x}^* \in \mathbb{R}^n$ be an existing local optimal solution. Assuming that the given convex optimization problem $f(\boldsymbol{x})$ is feasible, then there exists ε such that

$$f(\boldsymbol{x}^*) = \inf\{f(\boldsymbol{x}):\ g_i(\boldsymbol{x}) \le 0,\ i = 1,\ \cdots,\ m;\ h_j(\boldsymbol{x}) = 0,\ j = 1,\ \cdots,\ p;\ \left\|\boldsymbol{x} - \boldsymbol{x}^*\right\| \le \varepsilon\}\ . \quad (96)$$

Suppose that $\boldsymbol{x}^*$ is not globally optimal. Then there exists a feasible $\boldsymbol{y}$ so that $f(\boldsymbol{y}) < f(\boldsymbol{x}^*)$, which implies that $\left\|\boldsymbol{y} - \boldsymbol{x}^*\right\| > \varepsilon$. Consider that a point $\boldsymbol{z}$ is given by

$$\boldsymbol{z} = (1-\theta)\boldsymbol{x}^* + \theta\boldsymbol{y}\ , \qquad 0 < \theta = \frac{\varepsilon}{2\left\|\boldsymbol{y} - \boldsymbol{x}^*\right\|} < 1\ .$$

Then $\left\|\boldsymbol{z} - \boldsymbol{x}^*\right\| \le \varepsilon/2 < \varepsilon$ and by convexity of the objective function $f(\boldsymbol{x})$,

$$f(\boldsymbol{z}) \le (1-\theta) f(\boldsymbol{x}^*) + \theta f(\boldsymbol{y}) \le f(\boldsymbol{x}^*)\ ,$$

which contradicts Eq. (95). Therefore, if local optimal solutions of Eqs. (60–61) and (63–64) exist, they are also the unique global optimal solutions of the optimization problems, respectively.

5.2.4. Scheduling Algorithm

Scheduling of the NCS consists of two parts: (1) priority assignment and client arrangement in the NCS, and (2) scheduling algorithm implementation in the programming or protocol of the NCSs. In general, the second part can be achieved by introducing the existing scheduling algorithms in the real-time system to the NCS.

For the purposes of the priority assignment and plant arrangement, we assume under *Assumption 2* that there will be an idle bandwidth available for the initial bandwidth allocation. The rate of change of the system PIF in terms of the sampling

frequency can be obtained as $U_i(f_i) = \partial J_i / \partial(\overline{\tau}_i f_i)$. Initially, the controlled plants will be arranged by the following sequence,

$$U_1(f_1^{\min}) \le U_2(f_2^{\min}) \le \cdots \le U_N(f_N^{\min}). \tag{97}$$

In this preferred sequence, by changing the same amount of sampling frequency of each plant, Client *N* will yield the largest change in PIF so that the performance of the NCS can be improved in the fastest rate. The rate of change of the sampling frequency can be linear or constant. Consequently, Client *N* should be first given the idle network bandwidth if available. And as long as sufficient bandwidth is available in the NCS, the increment of BU for Client *N* will continue until the moment either that $U_N(f_N) = U_{N-1}(f_{N-1}^{\min})$, or $f_N = f_N^{\max}$, or $f_N = f_N^*$. Then BU of Plant *N* and *N*–1 will increase by maintaining $U_N(f_N) = U_{N-1}(f_{N-1})$ until the moment either that (1) $U_N(f_N) = U_{N-1}(f_{N-1}) = U_{N-2}(f_{N-2}^{\min})$, (2) $f_N = f_N^{\max}$ and $f_{N-1} = f_{N-1}^{\max}$, or (3) $f_N = f_N^*$ and $f_{N-1} = f_{N-1}^*$. This bandwidth allocation process will continue until the idle network bandwidth in the NCS is exhausted.

Note that the $U_i(f_i)$ consists of two parts, Ξ_i and Φ_i in the exponential approximation. Here, Ξ_i represents the decrement in the PIF per unit increment of the BU from the traditional digital design perspective, and Φ_i, the increment in the PIF per unit increment of the BU from the NCS design perspective, respectively. From Section 5.1.1, Eq. (58) is a monotonically decreasing function of f_i, and Eq. (59) is a monotonically increasing function of f_i. Recall Eq. (78) so that we have

$$\Xi_i e^{-\beta_i f_i} + \Phi_i e^{\delta_i f_i} + \lambda \overline{\tau}_i = 0, \quad i = l+1, \ \cdots, \ N. \tag{98}$$

For each iteration, λ and $\overline{\tau}_i$ are given from Eq. (79) and the timestamp calculation of Client, respectively. Hence, we can consider $\lambda \overline{\tau}_i$ as an offset of Eq. (98). If the offset is being eliminate from Eq. (98), one can have

$$\Xi_i e^{-\beta_i f_i} + \Phi_i e^{\delta_i f_i} = 0, \quad i = l+1, \ \cdots, \ N. \tag{99}$$

From the properties of exponential functions, $e^{-\beta_i f_i} < 1$ and $e^{\delta_i f_i} > 1$, respectively. Therefore, for non-trivial sampling frequency $f_i \neq 0$,

$$\left| \frac{\Xi_i}{\Phi_i} \right| = \left| \frac{e^{\delta_i f_i}}{e^{-\beta_i f_i}} \right| > 1, \quad i = l+1, \ \cdots, \ N. \tag{100}$$

From Eq. (95), we will have $|\Xi_i| > |\Phi_i|$ to balance the exponential functions in Eqs. (71) and (78) for each client in general. This indicates that the NCS is more sensitive with respect to Ξ_i than Φ_i. In Section 5.3, we will see that $|\Xi_i| \gg |\Phi_i|$ in simulation and experiments. Therefore, we will apply Ξ_i as the primary parameter and Φ_i as the complementary parameter for the scheduling.

The calculation of the network bandwidth saturation threshold *B* of an NCS depends on the scheduling algorithms. There exist several scheduling algorithms for real-time systems that could also be implemented in the NCS [62]. However, there is a significant difference between real-time system scheduling and NCS scheduling. Real-time system scheduling is able to put the tasks into a pre-emptive status based on their priority decided by the scheduling algorithms. But when a data packet is transmitted in the network, the controller will be unable to suspend the data packet although there

might be higher-priority tasks in the NCS. Therefore, only real-time non-preemptive scheduling algorithms can be applied to the NCS such as non-preemptive RM and EDF.

The flow chart of the scheduling algorithm of the NCS is illustrated in Fig. 51. The network bandwidth saturation threshold B of an RM scheduling defined in Fig. 51 is the ratio of the smallest task period over the largest task period in the system [89]. In an NCS, the task period can be assumed equal to the sampling period of Client. For instance, arranging the sampling period of a Client in an ascending order as $h_1 \le h_2 \le \cdots \le h_N$, then $B = h_1 / h_N$. With above analysis and discussions, the scheduling algorithm with the proposed NCS PIF approximations is as follows.

Algorithm 2: The scheduling algorithm of the proposed PIF approximations is

(1) Decides which scheduling algorithm will be implemented on the NCS. If RM is chosen, then $B = h_1 / h_N$; if EDF is chosen, then $B = 1$.

(2) Choose the approximations for the NCS. Based on the network bandwidth saturation threshold B from step (1), the optimal sampling frequencies of the given NCS are calculated from Eqs. (66–67) and (81–82) for the exponential and quadratic approximations, respectively.

(3) Arrange the clients in the order of Eq. (97) for the sampling frequencies update.

(4) Update the sampling frequency of each client until the optimal sampling frequency is achieved or the idle network bandwidth is exhausted.

5.3. SIMULATION AND EXPERIMENTS

In this section, the simulation and experimental results are presented to demonstrate the effectiveness of the proposed approximation methods and their

scheduling performances. Four DC-motor speed-control systems were set up as the test bed for experimental verification as shown in Fig. 13.

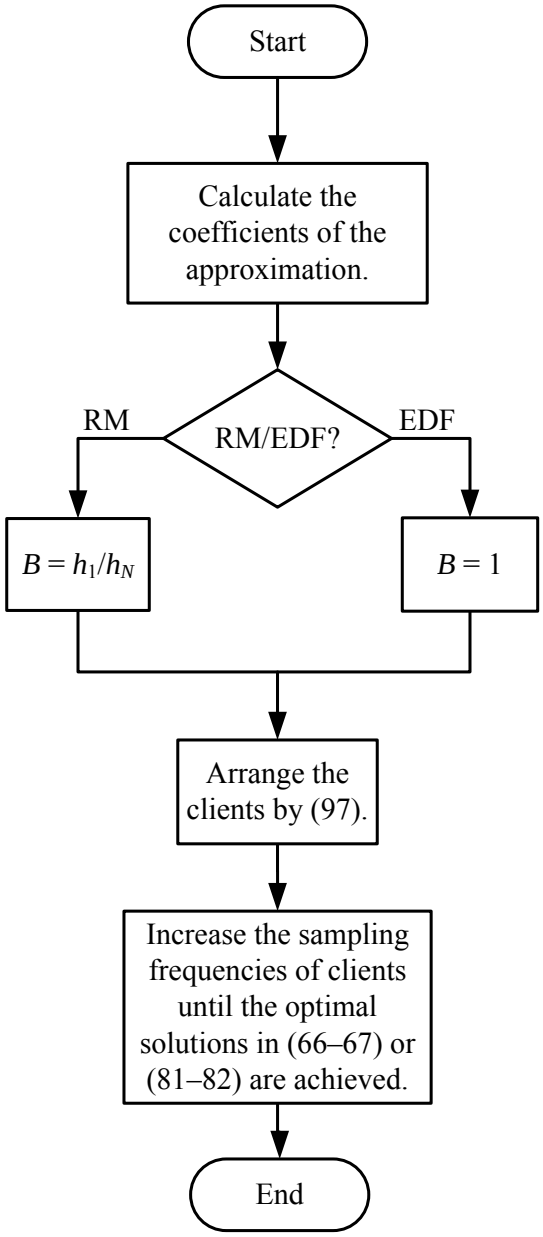


Fig. 51. A flow chart of the proposed optimal bandwidth allocation and scheduling algorithm of the NCS

The objective of this experiment is to control the speed of a DC motor over the LAN. The transfer functions of the DC motor and the PI controller are defined by Eqs. (10–11). Again, the reference speed was set to be 10 rps. The minimum sampling frequency without causing instability of each DC motor is 65 Hz. The average of total time delay of the DC motor speed-control system is measured to be 1.360 ms. The network protocol is UDP.

5.3.1. Simulation Results

In order to verify the scheduling algorithm proposed in Section 5.2, the simulation result is presented here first. The simulation setup contains four independent DC motor speed-control systems as given in Eqs. (10–11). To make this simulation closer to practice, the network-induced time delays are also considered. A time-delay experiment was performed for 20,000 iterations with Client 2 with a 3-ms sampling period as before. The average and the standard deviation of the time delays in the network are 0.5034 ms and 0.0414 ms, respectively. The data-packet processing time is not included here. The simulation in this section was conducted by the TrueTime toolbox in Matlab [84]. Ethernet is chosen as the network protocol and its transmission rate is 100 Mbps.

When all the DC motors are operated at the minimum sampling frequency of 65 Hz, the BU of each DC motor is 0.0884, and the total BU of the NCS is 35.36%. With either the EDF or RM scheduling algorithm, $B = 1$, and Assumption 2 is justified. Figure 52 shows the simulation of a single DC motor and its PIFs with the exponential and quadratic approximations are as follows

$$J_i = 930.9e^{-0.0315 f_i} + 0.006e^{0.0280 f_i}, \tag{101}$$

$$J_i = 0.0288 f_i^2 - 16.0297 f_i + 2776.6. \tag{102}$$

The sampling periods of the simulation were varied from 2.2 to 15.2 ms with a step of 0.2 ms. In each individual sampling period, the simulation ran for 20,000 control iterations.

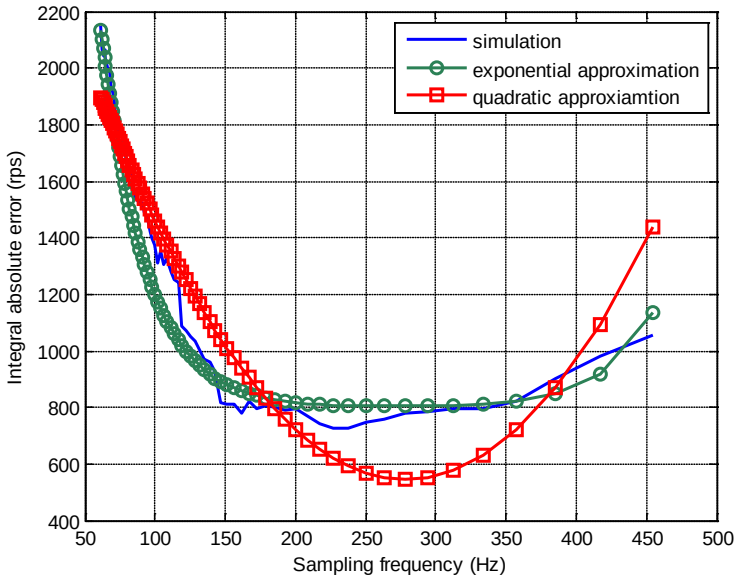


Fig. 52. DIAE vs. sampling frequencies of the simulation, exponential approximation, and quadratic approximation

For the exponential approximation, we have l = 1 from Eq. (68), where l is the smallest number of the controlled plants that have the minimum sampling frequency as defined in Eq. (68). Similarly, for the quadratic approximation, l = 1. Based on

Theorems 5 and 6, the optimal sampling frequency of each DC motor is given in Table 8. Note that the different selection of the weights will lead to different optimal solutions listed in Table 8. Here, for the four identical DC motors, we chose the weights based on the priority that we expected.

Table 8. Optimal sampling frequencies of the simulation

Client	# of DC motors	ω_i	Exponential Approximation [Hz]	Quadratic Approximation [Hz]
2	1	1	65	65
4	2	2	124.4	121.2
5	3	4	240.2	233.4
6	4	5	304.8	298.6

Figure 53 shows the profile of the sampling-frequency and BU changes for each DC motor during the simulation. Here, we adopted a linear changing rate for the sampling frequencies. Under Assumption 1, the time delay of each sampling frequency is assumed to be a constant. Hence, the BU changes can also be illustrated in the figure as the sampling frequency in Fig. 53.

From Table 8 and Eq. (55), the total BUs of the exponential and quadratic approximations are 99.87% and 97.67%, respectively. The parameters in the digital controller depended on the sampling frequency and were adjusted automatically as the sampling frequency changed.

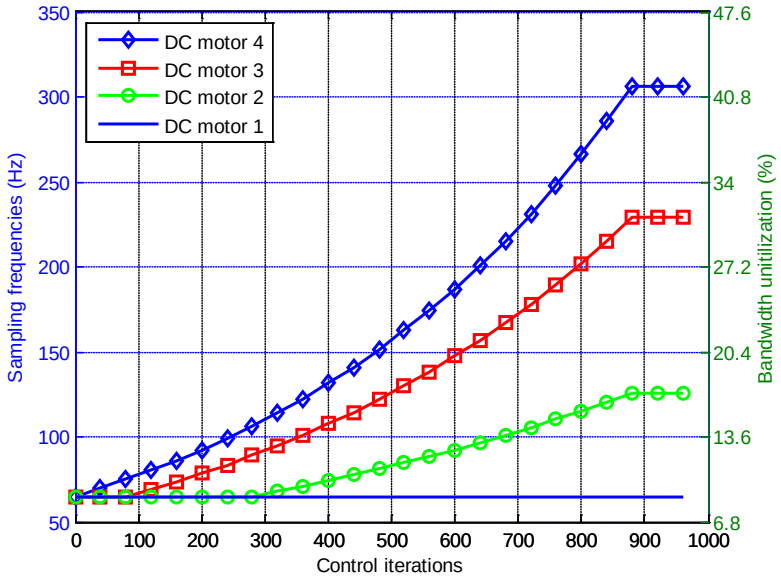


Fig. 53. Profile of the sampling-frequency and BU changes for each DC motor during the simulation

In Fig. 53, as the control iteration increased, the sampling frequency of each DC motor approached their optimal values listed in Table 8. After the optimal sampling frequencies were achieved, the DC motors kept the sampling frequencies for the rest of the simulations. Therefore, the total BU of the simulation may not be exactly the same as the ones calculated from Table 8, however, it will be very close to the optimal value eventually. From Fig. 53, the final total BU is 98.75%. Note that the sampling frequencies of DC motor 2, 3, and 4 in Fig. 53 are not changed in the beginning of the program. These delays are caused by the scheduling algorithm in Eq. (95). The clients will run at their minimum sampling frequencies until the scheduling condition in Eq. (95) is satisfied. Figure 54 shows the accumulated total cost of performance J of the simulation, the exponential approximation, and the quadratic approximation,

respectively. As the control iteration increases, the sampling frequency of each DC motor is approaching their optimal values listed in Table 8 and the total cost of the performance J is decreasing.

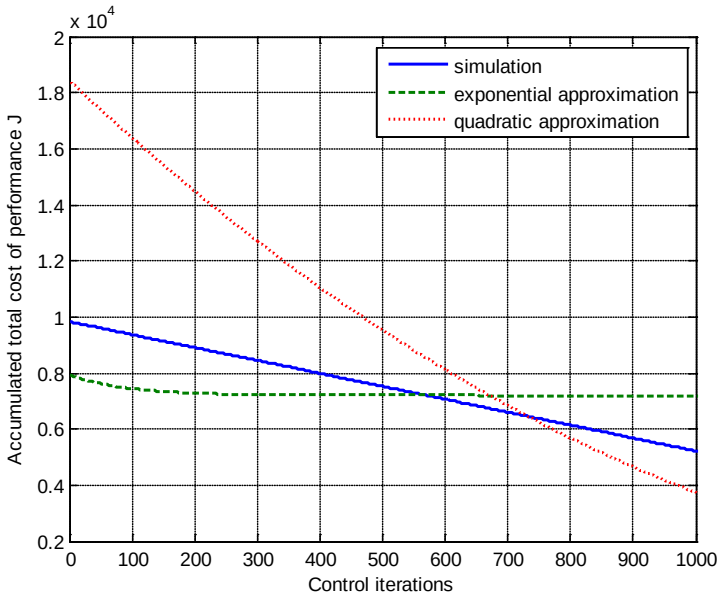


Fig. 54. Accumulated total cost of performance J of the simulation, exponential approximation, and quadratic approximation

5.3.2. Experimental Results without Reserved Bandwidth

The relation of the DIAE versus the sampling frequency and the time delay in experiments and with the exponential approximation are given in Figs. 55 and 56. Five sets of experiments were conducted under the same network conditions. In each individual sampling period, the experiments ran for 20,000 control iterations.

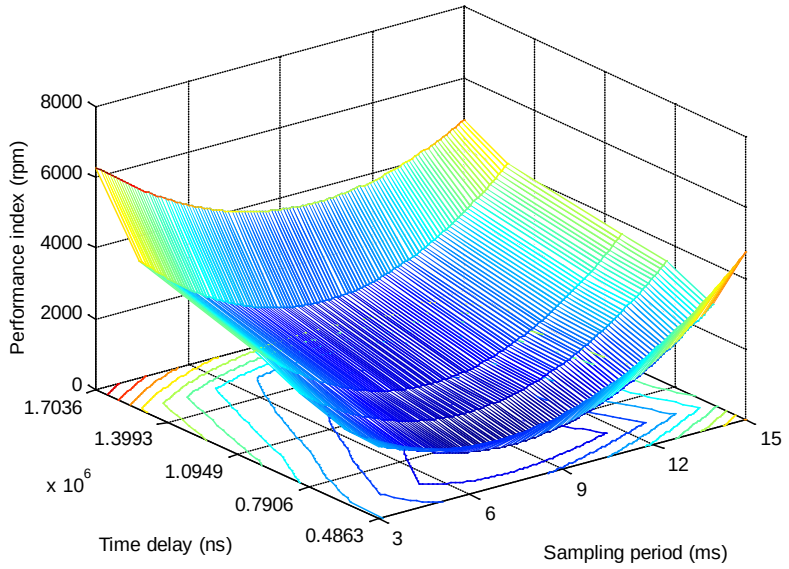


Fig. 55. DIAE vs. sampling period and time delay in experiments

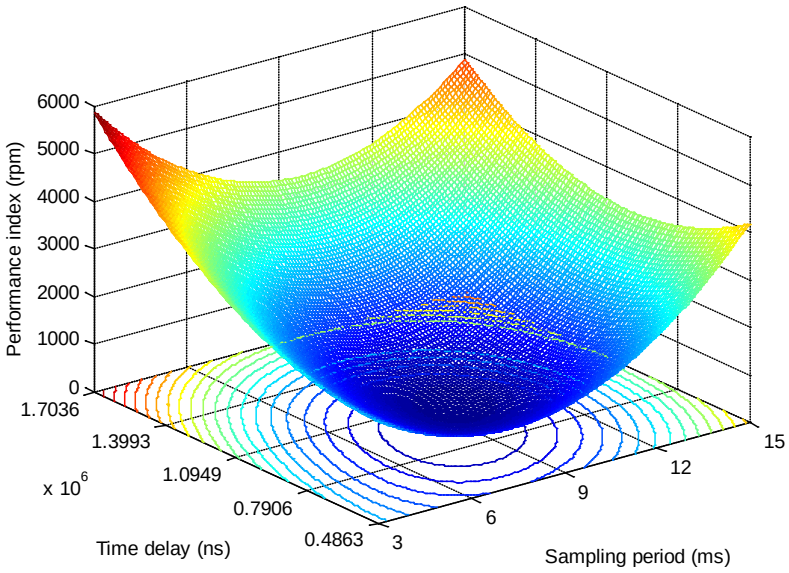


Fig. 56. DIAE vs. sampling period and time delay with the exponential approximation

Note that the time delays induced by the network in an NCS are not directly controllable variables. Extra time delays are manually inserted to the experiments. The curves of Fig. 55 are not smooth along either the sampling-period axis or the time-delay axis. This non-smoothness results from the uncertainties of the NCS, which makes the design of an NCS a challenge. Figure 56 shows the approximated exponential PIF generated by curve fitting. Compare Figs. 55 and 56, the trend of the performance of an NCS can be faithfully approximated by the exponential PIF.

The relation of the DIAE versus the sampling frequencies of the experiments and exponential/quadratic approximations is given in Fig. 57. Figure 57 shows the experimental results of a single DC motor and its exponential and quadratic approximations. From Fig. 57, the DIAE of the DC motor was quite large in the lower sampling-frequency range because the DC motor could not have adequately frequent control inputs from the controller to maintain the system performance. As the sampling frequency increased, the DIAE of the DC motor decreased. In the higher sampling-frequency range, the DIAE of the DC motor increased again because the number of data packets in the network increased as the sampling frequency kept increasing. The large number of the data packets would bring longer time delays into the NCS or even packet losses so that the performance of the DC motor could be degraded. Note that there are a few discrepancies among the five sets of experiments in the high sampling-frequency range. Because of the large number of the data packets in the network in the high sampling-frequency range, any possible disturbances or irrelevant data-packet

transmissions from other network users may cause the NCS performance to be degraded. Therefore, the NCS is more sensitive at high sampling frequencies.

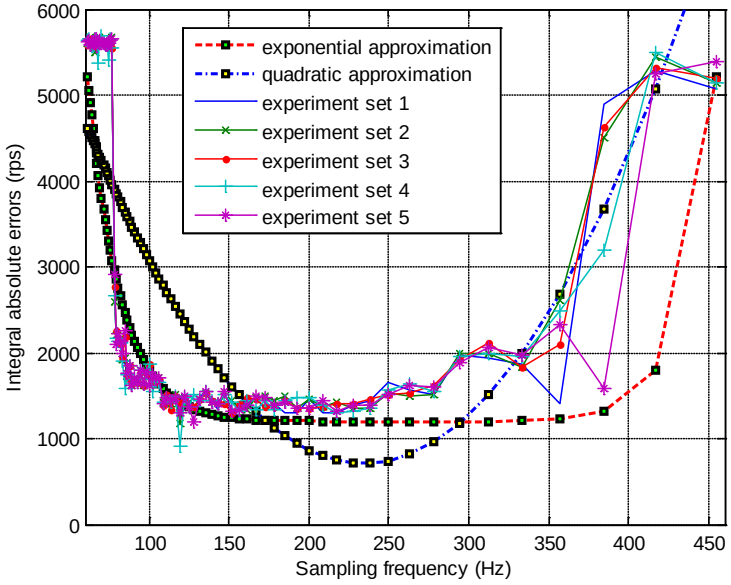


Fig. 57. DIAE vs. sampling frequencies of the experiments and exponential and quadratic approximations

Similarly, the PIFs with exponential and quadratic approximations are as follows

$$J_i = 8781e^{-0.05 f_i} + 0.001e^{0.0556 f_i}, \tag{103}$$

$$J_i = 0.1316 f_i^2 - 67.1258 f_i + 7941.49. \tag{104}$$

Although the simulations and the experiments were conducted based on the same system dynamics of the DC motors and the controller, the approximations of the simulation and experiments are different. This is because the experiments were conducted under a complicated network environment and had more uncertainties than the simulations. Our

LAN is shared with other non-NCS users when the experiments were conducted. Longer time delays and packet losses might have existed in the network in experiments. Similarly as in Section 5.3.2, we have l = 1 for both the exponential and quadratic approximations, respectively. Therefore, based on *Theorems 5* and *6*, the optimal sampling frequency of each DC motor is given in Table 9.

Table 9. Optimal sampling frequencies of the experiments

Client	DC motor	ω_i	Exponential Approximation [Hz]	Quadratic Approximation [Hz]
2	1	1	65	65
4	2	2	173.5	162.7
5	3	4	237.1	225.8
6	4	5	252.2	249.9

Figure 58 shows the profile of the sampling-frequency and BU changes for each DC motor during the experiments. Here, we adopted a linear changing rate for the sampling frequencies. From Table 9 and Eq. (55), the total BUs of the exponential and quadratic approximations are 98.98% and 95.66%, respectively. Similarly as the simulation, the sampling frequency were adjusted automatically by $f_{i+1} = cf_i$ where c is a constant rate of change. In Fig. 58, as the control iteration increased, the sampling frequency of each DC motor approached their optimal values listed in Table 9. The total BU of the experiments may not be exactly the same as the ones calculated from Table 9, however, it will be close to the optimal value eventually.

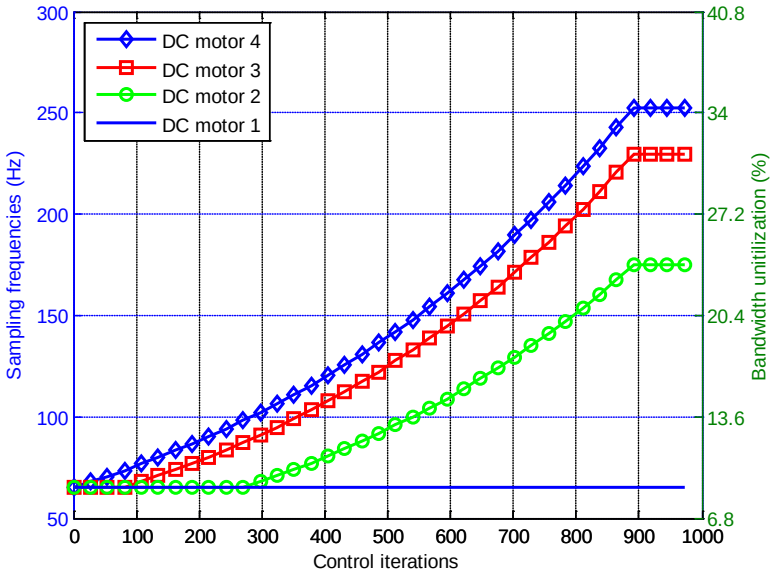


Fig. 58. Profile of the sampling-frequency and BU changes for each DC motor during the experiments

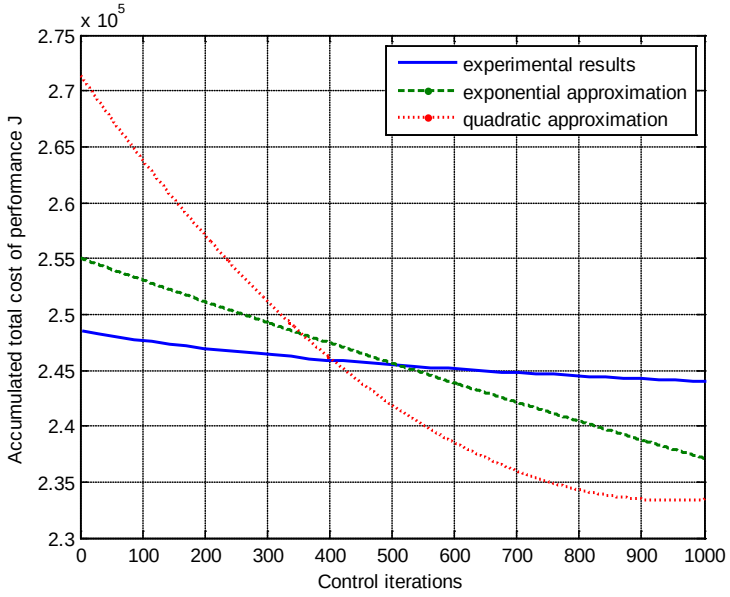


Fig. 59. Accumulated total cost of performance *J* of the experiments, exponential approximation, and quadratic approximation

From Fig. 58, the final total BU is 98.26%. Figure 59 shows the accumulated total cost of performance J of the experiment, the exponential approximation, and the quadratic approximation, respectively. As the control iteration increases, the sampling frequency of each DC motor is approaching their optimal values listed in Table 9 and the total cost of the performance J is decreasing.

5.3.3. Experimental Results with Reserved Bandwidth

In next experiment, Client 1, the ball maglev system, is introduced in to this experiment. As mention earlier in Section 3, it is an open-loop unstable system that has a fixed sampling frequency of 333 Hz. The average of total time delay $\overline{\tau}_1$ is measured to be 1.350 ms from the experiments. By Eq. (55), the BU of Client 1 is $b_1 = \overline{\tau}_1 f_1 = (1.350 \times 10^{-3} \text{ s}) \times (333 \text{ Hz}) = 44.96\%$. The total time delay of the ball maglev system and the DC motor speed-control system are very close because they are tested under the same network conditions and have the same size of data packets. With the reserved network bandwidth of Client 1, the available network bandwidth $\hat{B} = B - \sum_{j \in \mathbb{J}} \overline{\tau}_j f_j = 1 - 0.4496 = 55.04\%$, where $\mathbb{J} = \{1\}$, we have $l = 2$ for the exponential approximation and the quadratic approximation, respectively. The optimal sampling frequency of each DC motor with the reserved network bandwidth in the NCS is given in Table 10.

Figure 60 shows the profile of the sampling-frequency and BU changes for each DC motor during the experiments with the ball maglev system as a controlled plant that had a fixed sampling frequency. From Table 10 and Eq. (55), the total BUs of the

exponential and quadratic approximations are 54.26% and 54.04%, respectively, with the ball maglev system taking approximately 45% of the total BU. Similarly, the total BU of the experiments may not be exactly the same as the ones calculated from Table 10, however, it will get close to the optimal value eventually. From Fig. 60, the final total BU is 53.45%. Note that DC motors 1 and 2 have the same optimal sampling frequencies as in Table 10, so they are overlapped in Fig. 60.

Table 10. Optimal sampling frequencies of the experiments with the ball maglev system

Client	DC motor	ω_i	Exponential Approximation [Hz]	Quadratic Approximation [Hz]
2	1	1	65	65
4	2	2	65	65
5	3	4	117.68	119.35
6	4	5	151.35	148.02

Figure 61 shows the accumulated total cost of performance J of the experiment with the ball maglev system, the exponential approximation, and the quadratic approximation, respectively. As the control iteration increases, the sampling frequency of each DC motor is approaching their optimal values listed in Table 10 and the total cost of the performance J is decreasing.

From the simulation and experiments, we can see the exponential approximation represents the practices more closely compared to the quadratic approximation in the sense of the system performance. However, the quadratic approximation takes less

computational efforts to solve the optimization objective function and has a closed-form optimal solution. Both the exponential and quadratic approximations could find the optimal sampling frequencies that exhaust about 98% of the total network bandwidth available to the NCS with or without the fixed sampling-frequency plant, the ball maglev system in our experiments.

Statistic comparisons of the exponential and quadratic approximations are given in Table 11. These statistical values are based on the simulation and experimental results in Figs. 52 and 57. From Table 11, the accuracy of the exponential approximation is about 30% and 60% better than the quadratic approximation for simulation and experiments, respectively. Although the exponential approximation is more accurate when capturing the system performance of an NCS, it does not have an analytic closed-form optimal solution that can be implemented on line. In contrast to the exponential approximation, the quadratic one has a closed-form optimal solution to the objective PIF that can be implemented on line. Hence, the algorithm of bandwidth allocation and scheduling with quadratic approximation has a better efficiency in real-time operation. However, the accuracy of the quadratic approximation is less than the exponential approximation. Moreover, the exponential approximation gives an explicit measurement of the effects of high sampling frequency on the NCS whereas the quadratic approximation only indicates a coupled performance measurement of the NCS.

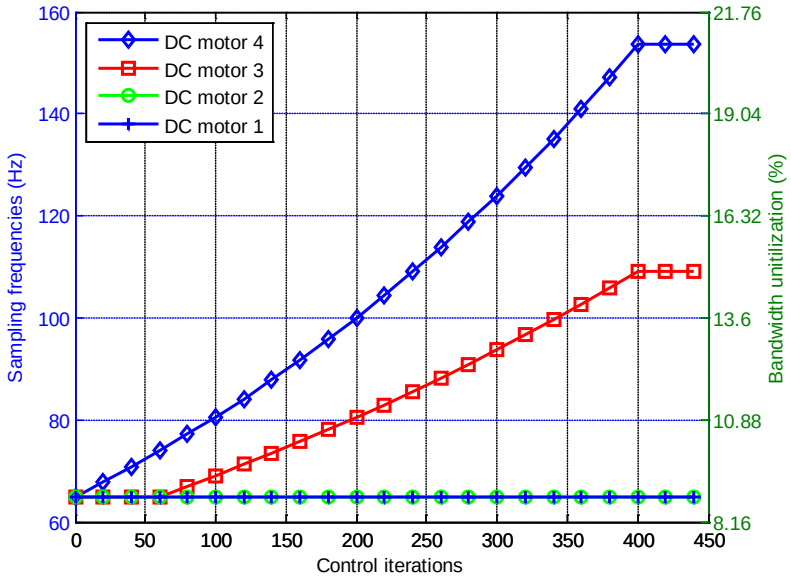


Fig. 60. Profile of the sampling-frequency and BU changes for each DC motor during the experiments with the ball maglev system

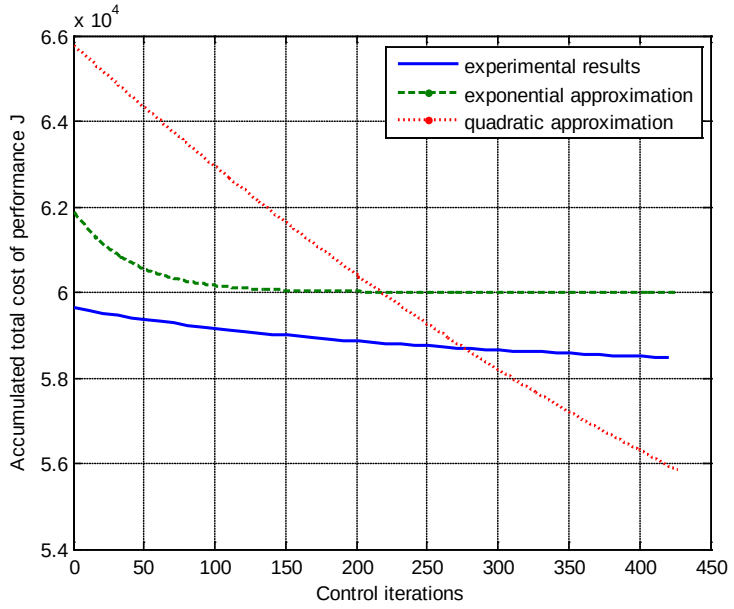


Fig. 61. Accumulated total cost of performance *J* of the experiments with the ball maglev system, exponential approximation, and quadratic approximation

Table 11. Statistic comparison of the exponential and quadratic approximations

		Exponential Approximation (rps)	Quadratic Approximation (rps)
Simulation	Mean	87.09	110.22
	Stdev	60.12	75.48
Experiments	Mean	395.42	1004.5
	Stdev	125.33	526.96

5.3.4. Experimental Results without Chosen Client Sequences

Note that the proposed approximations and algorithms are intend to minimize the system error while exhaust all the available network resources of an NCS. The approximations can also be applied to minimize the BU to obtain an optimal system performance. This can be achieved by removing the pre-set client sequence defined in Eq. (99). Then the optimal solution to the proposed PIF approximations from the KKT conditions will be optimal BU that leads to the minimum system errors. The following experiments show the optimal results of this case.

As in Section 5.3.2, when all the DC motors are operated at their minimum sampling frequency of 65 Hz, the BU of each DC motor is 8.84%, and the total BU of the NCS is 35.36%. The optimal sampling frequency of each DC motor with the EDF scheduling algorithm is given in Table 12 indicated by Case 1. The total BU is 78.24%. Table 12 also gives the sampling frequencies of each DC motor of various BU

combination cases. Cases 2 and 3 evenly distribute the 79% and 100% bandwidths to each DC motor. Note that Case 2 has the same total BU as Case 1.

Figure 62 shows the DIAE of the BU cases as given in Table 12. Note that Cases 1 and 2 have nearly same total bandwidths of the entire NCS. However, from Fig. 10, the DIAE of Case 1 is reduced by about 25% than that of Case 2. Although both Cases 2 and 3 evenly distribute the bandwidth to each DC motor, the DIAE of Case 3 is larger than that of Case 2. In these two cases, each DC motor has the same sampling frequency, so they send data packets to the controller simultaneously. Case 3 has nearly 100% bandwidth, so the scheduler and the controller are busy with data transmission and data calculation. It may induce longer waiting time to each data packet in the network, and its system performance is worse than Case 2. From the experiments, one can see that the exponential approximation could represent the practice closely in terms of the system performance and yield the optimal sampling frequency of the NCS.

Table 12. Sampling frequencies [Hz] of Cases

DC motor	ω_i	Case 1 (f_i^*)	Case 2	Case 3
1	1	116.27	145.22	183.82
2	2	132.55	145.22	183.82
3	4	145.10	145.22	183.82
4	5	181.37	145.22	183.82

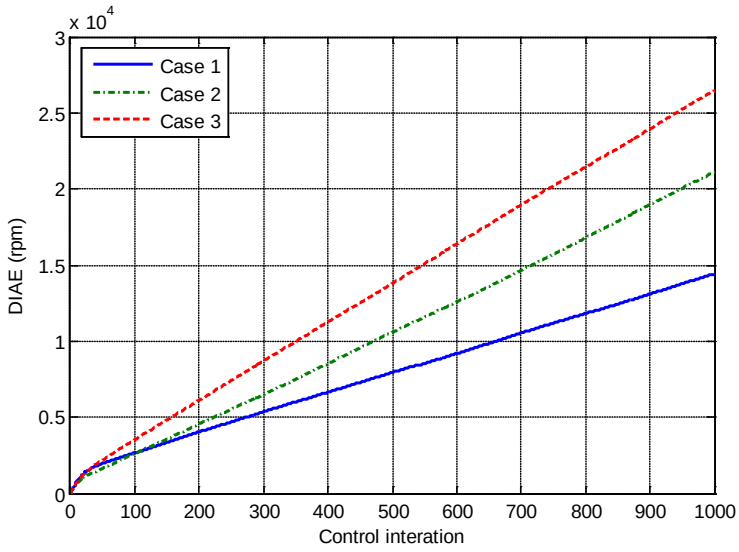


Fig. 62. DIAE in the experiments given in Table 12

5.4. SUMMARY

In this section, the optimal bandwidth allocation and scheduling of the NCS with nonlinear-programming techniques were investigated. The BU of each controlled plant was defined in terms of its sampling frequency. Two nonlinear approximations, exponential and quadratic, were formulated to describe the system performance governed by the DIAE versus the sampling frequencies. Based on the convexity of the proposed approximations, the optimal solution could be obtained from the nonlinear-programming perspective. The optimal sampling frequencies were obtained by solving the approximations with the KKT conditions. Within various network bandwidth saturation thresholds based on different real-time scheduling algorithms, the proposed approximations could find the optimal BU for each controlled plant in the NCS. Later in

the section, simulation and experimental results verified the effectiveness of the proposed approximation models. In both simulation and experiments, the total BU of the NCS could approach up to 98% of the total available network bandwidth. Therefore, the proposed approximations and the scheduling algorithms can maximize the BU so that the controlled plants can be scheduled along with the system PIFs being optimized. Experimental results also verified the effectiveness of the proposed approximation method to solve the NCS bandwidth allocation without chosen client sequences. Therefore, the proposed approximation can minimize the BU so that the plants can be scheduled along with the system PIFs being optimized.

6. CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

This dissertation presented the output feedback control and optimal bandwidth allocation of an NCS. This section will provide the conclusions of current research and several suggestions for future work.

6.1. CONCLUSIONS

An NCS is a control system that has sensors, actuators, and controllers geographically distributed and connected by communication media. The NCS has advantages of low cost, reduced system weight, easy installation and maintenance, high flexibility, etc. However, the communication media bring several challenges such as the network-induced time delays, packet losses, resource allocation, scheduling, etc. This dissertation mainly focused on the output feedback control and optimal bandwidth allocation of an NCS. It covered the design issues of the NCS including network-induced time delays, packet losses, resource allocation, and scheduling and could be used as guidance in the design of an NCS.

The NCS discussed in this dissertation included three different types of clients, the ball maglev system, the DC motor-speed control system, and the wireless robotic wheelchair system. These three systems represented the fast, medium, and slow dynamics in the entire NCS. We briefly discussed the capability and system performance of an NCS that has both wired and wireless clients, and also the capability of the NCS controlling various dynamic systems.

Later, this dissertation proposed an output feedback method for the stabilization and control of the NCS with random time delays and packet losses. This output feedback control considered the time delays and packet losses in both the sensor-to-controller and controller-to-actuator links. The random time delays were modeled with two time-homogeneous Markov chains, and the packet losses with Dirac delta functions, respectively. Then an asymptotic mean-square stability criterion for the NCS was obtained in terms of a Lyapunov function and a set of LMIs with matrix constraints based on the time-delay and packet-loss models. An algorithm implementation of the stability criterion was also presented later to verify the effectiveness of the proposed output feedback controller design method. The experimental results collected from Client 2 demonstrated the feasibility and effectiveness of the proposed method. It enhanced the system performance with and without packet losses compared to a conventional control algorithm. The NCS could track the reference command faithfully with the proposed method when random time delays and packet losses existed in the links whereas the NCS failed to track the reference command with a conventional control algorithm.

Lastly, this dissertation investigated the optimal bandwidth allocation and scheduling of the NCS with the nonlinear-programming techniques. The BU of each client was defined in terms of its sampling frequency. Two nonlinear approximations, exponential and quadratic approximations, were formulated to describe the system performance governed by the DIAE versus the sampling frequencies. The optimal sampling frequencies were obtained by solving the approximations with the KKT

conditions. A simple scheduling algorithm was proposed to perform the optimal bandwidth allocation calculated from the two approximations. The scheduling algorithm was based on the changing rate of the PIFs versus the changing rate of the sampling frequencies of each client. Later, simulation and experimental results verified the effectiveness of the proposed approximations and the bandwidth allocation and scheduling algorithms. In both the simulation and experiments, the two approximations could maximize the total BU of the NCS up to about 98% of the total available network bandwidth.

6.2. SUGGESTIONS FOR FUTURE WORK

This dissertation provides theoretical foundations for the future research efforts in the NCS design and performance analysis. In this section, a few possible further research directions are explored.

For the NCS, the network-induced time delays include the data processing time is a significant factor that affects the total time delays. So are the possible data-packet losses in the network. This dissertation modeled the time delays with Markov chains and the packet losses with Dirac Delta models. However, the NCS designer needs to have a good knowledge of the statistics of the time delays and packet losses prior to performing the controller design procedure. In this dissertation, several experiments were conducted to measure the statistics of the time delays to set up the Markov chain probability transition matrix before fully designing the controller. From the computer science perspective, the behaviors of the data packets in the network can be modeled a with Bernoulli or Poison process. Doing such, the NCS designer may not have to measure the

statistics of the time delays from the experiments but calculate them from the Bernoulli or Poison process. Therefore, the output feedback design process could be further developed by including the Bernoulli or Poison process.

The packet-loss model presented in this dissertation considered both the single packet losses and consecutive packets losses during data-packet transmission. An AR model was applied to predict the lost data packet in the network. An AR model is the simplest prediction model in the NCS design. However, a more robust prediction model could be applied to enhance the performance.

Optimal network scheduling of the NCS also deserves more efforts. The scheduling algorithm proposed in this dissertation based on the system performance changing rate versus the sampling frequency. The scheduling depended on a certain sequence of the clients set up prior to the experiments based on the various PIFs. However, a more advanced dynamic scheduling algorithm may enhance the NCS performance. This dynamic scheduling algorithm should consider the system performance on a real-time basis and be able to assign different scheduling schemes to different clients.

REFERENCES

[1] P. Ogren, E. Fiorelli, and N. E. Leonard, "Cooperative control of mobile sensor networks: Adaptive gradient climbing in a distributed environment," *IEEE Trans. on Automatic Control*, vol. 49, no. 8, pp. 1292–1302, Aug. 2004.

[2] C. Meng, T. Wang, W. Chou, S. Luan, Y. Zhang, and Z. Tian, "Remote surgery case: Robot-assisted teleneurosurgery," in *Proc. IEEE Int. Conf. of Robot and Automation*, vol. 1, pp. 819–823, Apr. 2004.

[3] J. P. Hespanha, M. L. McLaughlin, and G. Sukhatme, "Haptic collaboration over the Internet," in *Proc. 5th Phantom Users Group Workshop*, pp. 9–13, Oct. 2000.

[4] K. Hikichi, H. Morino, I. Arimoto, K. Sezaki, and Y. Yasuda, "The evaluation of delay jitter for haptics collaboration over the Internet," in *Proc. IEEE Global Telecomm Conference*, vol. 2, pp. 1492–1496, Nov. 2000.

[5] P. Seiler and R. Sengupta, "Analysis of communication losses in vehicle control problems," in *Proc. American Control Conf.*, vol. 2, pp. 1491–1496, Jun. 2001.

[6] P. Seiler and R. Sengupta, "An H_∞ approach to networked control," *IEEE Trans. on Automatic Control*, vol. 50, no. 3, pp. 356–364, Mar. 2005.

[7] R. A. Gupta and M.-Y. Chow, "Networked control system: Overview and research trends," *IEEE Trans. on Industrial Electronics*, vol. 57, no. 7, pp. 2527–2535, Jul. 2010.

[8] Y. Halevi and A. Ray, "Integrated communication and control systems: Part I–Analysis," *J. Dyn. Syst. Meas. Cont.*, vol. 110, no. 4, pp. 367–373, Dec. 1988.

[9] Y. Halevi and A. Ray, "Integrated communication and control systems: Part II–Design consideration," *J. Dyn. Syst. Meas. Cont.*, vol. 110, no. 4, Dec., pp. 374–381, Dec. 1988.

[10] S. Lichiardopol, "A survey on teleoperation," *DCT report*, Dec. 2007.

[11] M. K. Habib, "Collaborative teleoperation design requirements and development issues," in *Proc. 26th Annual Conf. of IEEE on Industrial Electronics Society (IECON'00)*, vol. 1, pp. 19–27. Oct. 2000.

[12] G. Hirzinger, B. Brunner, J. Dietrich, and J. Heindl, "ROTEX – The first remotely controlled robot in space," in *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 3, pp.2604–2611, May 1994.

[13] K. Goldberg, "The Mercury project: A feasibility study for Internet robots," *IEEE Robotics and Automation Magazine*, vol. 7, no. 1, pp. 35–40, Mar. 2000.

[14] R. C. Luo, J. H. Tzou, and Y. C. Chang, "Desktop rapid prototyping system with supervisory control and monitoring through Internet," *IEEE/ASME Trans. on Mechatronics*, vol. 6, no. 4, pp. 399–409, Dec. 2001.

[15] C. E. Garcia, R. Carelli, J. F. Postigo, and C. Soria, "Supervisory control for a telerobotic System: A hybrid control approach," *Control Engineering Practice*, vol. 11, no. 7, pp. 805–817, Jul. 2003.

[16] P. G. Backes, K. S. Tso, and G. K. Tharp, "Mars pathfinder mission Internet based operations using WITS," in *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 1, pp. 284–291, May 1998.

[17] K. Brady and T. J. Tarn, "Internet-based remote teleoperation," in *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 1, pp. 65–70, May 1998.

[18] A. Srivatsava, *Distributed Real-time Control via Internet*, M.S. Thesis, Texas A&M University, May 2003.

[19] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 138–162, Jan. 2007.

[20] Y. Tipsuwan and M.-Y. Chow, "Control methodologies in networked control systems," *Control Engineering Practice*, vol. 11, no. 10, pp. 1099–1111, Nov. 2003.

[21] J. W. Overstreet and A. Tzes, "An Internet-based real-time control engineering laboratory," *IEEE Control System Magazine*, vol. 19, no. 5, pp. 19–34, Oct. 1999.

[22] Y. Tipsuwan and M.-Y. Chow, "Network-based controller adaptation based on QoS negotiation and deterioration," in *Proc. 27th Annual Conf. of IEEE on Industrial Electronics Society (IECON'01)*, vol. 3, pp. 1094–1099, Nov. 2001.

[23] Y. Tipsuwan and M.-Y. Chow, "Gain adaptation of networked mobile robot to compensate QoS deterioration," in *Proc. 28th Annual Conf. of IEEE on Industrial Electronics Society (IECON'02)*, vol. 4, pp. 3146–3151, Nov. 2002.

[24] T.-J. Tarn and N. Xi, "Planning and control of Internet-based teleoperation," in *Proc. of SPIE: Telemanipulator and telepresence technologies V*, vol. 3524, pp. 189–193, Dec. 1998.

[25] F.-L. Lian, *Analysis, Design, Modeling, and Control of Networked Control Systems*, Ph.D. Dissertation, University of Michigan, 2001.

[26] K. Ji and W.-J. Kim, "Optimal bandwidth allocation and QoS-adaptive control co-design for networked control systems," *Int. Journal of Control, Automation, and Systems*, vol. 6, no. 4, pp. 596–606, Aug. 2008.

[27] F.-L. Lian, J. Moyne, and D. Tilbury, "Network design consideration for distributed control systems," *IEEE Trans. on Control Systems Technology*, vol. 10, no. 2, pp. 297–306, Mar. 2002.

[28] K. Ji and W.-J. Kim, "Real-time control of networked control systems via Internet," *Int. Journal of Control, Automation, and Systems*, vol. 3, no. 4, pp. 591–600, Dec. 2005.

[29] W. Zhang, M. S. Branicky, and S. M. Phillips, "Stability of networked control system," *IEEE Control Systems Magazine*, vol. 21, no. 2, pp. 84–99, Feb. 2001.

[30] H. Lin, G. Zhai, and P. J. Antsaklis, "Robust stability and disturbance attenuation analysis of a class of networked control systems," in *Proc. 42nd IEEE Conf. on Decision and Control*, vol. 2, pp. 1182–1187, Dec. 2003.

[31] J. Nilsson, *Real-time Control Systems with Delays*, Ph.D. Dissertation, Lund Institute of Technology, Sweden, 1998.

[32] J. Nilsson, B. Bernhardsson, and B. Wittenmark, "Stochastic analysis and control of real-time systems with random time delays," *Automatica*, vol. 34, no. 1, pp. 57–64, Jan. 1998.

[33] S. Hu and W. Zhu, "Stochastic optimal control and analysis of stability of networked control system with long delay," *Automatica*, vol. 39, pp. 1877–1884, Nov. 2003.

[34] L. Xiao, A. Hassibi, and J. P. How, "Control with random communication delays via a discrete-time jump system approach," in *Proc. American Control Conference*, vol. 3, pp. 2199–2204, Jun. 2000.

[35] L. Zhang, Y. Shi, T. Chen, and B. Huang, "A new method for stabilization of networked control systems with random delays," *IEEE Trans. on Automatic Control*, vol. 50, no. 8, pp. 1177–1181, Aug. 2005.

[36] Y. Shi and B. Yu, "Output feedback stabilization of networked control systems with random delays modeled by Markov chains," *IEEE Trans. on Automatic Control*, vol. 54, no. 7, pp. 1668–1674, Jul. 2009.

[37] Y. Shi and B. Yu, "Robust mixed H_2/H_∞ control of networked control systems with random time delays in both forward and backward communication links," *Automatica*, vol. 47, no. 4, pp. 754–760, Apr. 2011.

[38] S.-L. Hu, J.-L. Liu, and Z.-P. Du, "Stabilization of discrete-time networked control systems with partly known transmission delay: a new augmentation approach," *Int. Journal of Control, Automation, and Systems*, vol. 9, no. 6, pp. 1080–1085, Dec. 2011.

[39] F. Yang, W. Wang, Y. Niu, and Y. Li, "Observer-based H_∞ control for networked systems with consecutive packet delays and losses," *Int. Journal of Control, Automation, and Systems*, vol. 8, no. 4, pp. 769–775, Aug. 2010.

[40] T. Jia, Y. Niu, and X. Wang, "H_∞ control for networked systems with data packet dropout," *Int. Journal of Control, Automation, and Systems*, vol. 8, no. 2, pp. 198–203, Apr. 2010.

[41] R. Krtolica, U. Ozguner, H. Chan, H. Goktas, J. Winkelman, and M. Liubakka, "Stability of linear feedback systems with random communication delays," *Int. Journal of Control*, vol. 59, no. 4, pp. 925–953, Apr. 1994.

[42] X. Ye, S. Liu, and P. X. Liu, "Modeling and stabilisation of networked control system with packet loss and time-varying delays," *IET Control Theory and Application*, vol. 6, no. 6, pp. 1094–1100, Jun. 2010.

[43] J. Xiong and J. Lam, "Stabilization of linear systems over networks with bounded packet loss," *Automatica*, vol. 43, no. 1, pp. 80–87, Jan. 2007.

[44] G. P. Liu, J. X. Mu, D. Rees, and S. C. Chai, "Design and stability analysis of networked control systems with random communication time delay using the modified MPC," *Int. Journal of Control*, vol. 79, no. 4, pp. 288–297, Apr. 2006.

[45] L. Schenato, "Optimal estimation in networked control systems subject to random delay and packet drop," *IEEE Trans. on Automatic Control*, vol. 53, no. 5, pp. 1311–1317, Jun. 2008.

[46] K. E. Arzen, A. Cervin, J. Eker, and L. Sha, "An introduction to control and scheduling co-design," in *Proc. 39th IEEE Conf. on Decision and Control*, vol. 5, pp 4865–4870, Dec. 2000.

[47] A. T. Al-Hammouri, M. S. Branicky, V. Liberatore, and S. M. Phillips, "Decentralized and dynamic bandwidth allocation in networked control

systems," in *Proc. 20th Int. Parallel and Distributed Processing Symposium, (IPDPS'06)*, pp. 25–29, Apr. 2006.

[48] M. Velasco, J. M. Fuertes, C. Lin, P. Martí, and S. Brandt, "A control approach to bandwidth management in networked control systems," in *Proc. 30th Annual Conf. of IEEE on Industrial Electronics Society (IECON'04)*, vol. 3, pp. 2343–2348, Nov. 2004.

[49] W. S. Wong and R. W. Brockett, "Systems with finite communication bandwidth constraints – Part I: State estimation problems," *IEEE Trans. on Automatic Control*, vol. 42, no. 9, pp. 1294–1299, Sep. 1997.

[50] W. S. Wong and R. W. Brockett, "Systems with finite communication bandwidth constraints – Part II: Stabilization with Limited Information Feedback," *IEEE Trans. on Automatic Control*, vol. 44, no. 5, pp. 1049–1053, May 1999.

[51] P. Martí, C. Lin, S. Brandt, M. Velasco, and J. M. Fuertes, "Optimal state feedback based resources allocation for resources-constrained control tasks," in *Proc. 25th IEEE Real-Time Operating Systems Symposium*, pp. 161–172, Dec. 2004.

[52] R. Castané, P. Martí, M. Velasco, A. Cervin, and D. Henriksson, "Resources management for control tasks based on the transient dynamics closed-loop systems," in *Proc. 18th Euro. Conf. on Real-Time Operating Systems (ECRTS'06)*, pp. 172–182, Jul. 2006.

[53] P. Belzarena, A. Ferragut, and F. Paganini, "Network bandwidth allocation via distributed auctions with time reservations," in *Proc. IEEE INFOCOM'09*, pp. 2816–2820, Apr. 2009.

[54] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "On task schedulability in real-time control systems," in *Proc. 17th IEEE Real-Time Operating Systems Symposium*, pp. 13–21, Dec. 1996.

[55] H. S. Park, Y. H. Kim, D.-S. Kim, and W. H. Kwon, "A scheduling method for network-based control systems," *IEEE Trans. on Control Systems Technology*, vol. 10, no. 3, pp. 318–330, May 2002.

[56] M. S. Branicky, S. M. Phillips, and W. Zhang, "Scheduling and feedback co-design for networked control systems," in *Proc. 41st IEEE Conf. on Decision and Control*, vol. 2, pp. 1211–1217, Dec. 2002.

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

[58] G. Weiss, S. Fischmeister, M. Anand, and R. Alur, "Specification and analysis of network resources requirements of control systems," in *Proc. 12th Int. Conf. on Hybrid Systems: Computation and Control (HSCC)*, pp. 381–395, Apr. 2009.

[59] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "Trade-off analysis of real-time control performance and schedulability," *Real-Time Systems*, vol. 21, no. 3, pp. 199–217, Nov. 2001.

[60] D.-S. Kim, D.-H. Choi, and P. Mohapatra, "Real-time scheduling method for networked discrete control systems," *Control Engineering Practice*, vol. 17, no. 5, pp. 564–570, May 2009.

[61] D.-S. Kim, J. Jeon, and P. Mohapatra, "Scheduling of wireless control networks based on IEEE 802.15.4 networks: Mixed traffic environment," *Control Engineering Practice*, vol. 19, no. 10, pp. 1223–1230, Oct. 2011.

[62] J. W. S. Liu, *Real-Time Systems*, Prentice Hall, Upper Saddle River, NJ, 2000.

[63] S. C. Paschall II, *Design, Fabrication, and Control of A Single Actuator Magnetic Levitation System*, Senior Honors Thesis, Texas A&M University, College Station, TX, May 2002.

[64] M. H. Lee, *Real-Time Networked Control with Multiple Clients*, M.S. Thesis, Texas A&M University, College Station, TX, Aug. 2009.

[65] P. Hsieh, *Autonomous Robotic Wheelchair with Collision-Avoidance Navigation*, M.S. Thesis, Texas A&M University, College Station, TX, Aug. 2008.

[66] P. Mantegazza, DIAPM RTAI – Real-time application, [Online] Available: http://www.rtai.org.

[67] D. Schleef, Linux control and measurement device interface, [Online] Available: http://www.comedi.org.

[68] Samba, [Online] Available: http://www.samba.org.

[69] N. J. Ploplys, P. A. Kawka, and A. G. Alleyne, "Closed-loop control over wireless network," *IEEE Control System Magazine*, vol. 24, no. 3, pp. 57–71, Jun. 2004.

[70] I. Lopez, J. L. Piovesan, C. T. Abdallah, D. Lee, O. Martinez, and M. Spong, "Practical issues in networked control systems," in *Proc. of American Control Conf.*, pp. 4201–4206, Jun. 2006.

[71] G. F. Franklin, J. D. Powell, and M. L. Workman, *Digital Control of Dynamic Systems*, 6th Ed., Prentice Hall, Upper Saddle River, NJ, 2009.

[72] E. Çinlar, *Introduction to Stochastic Process*, Prentice Hall, Upper Saddle River, NJ, 1997.

[73] S. C. Smith and P. Seiler, "Estimation with lossy measurements: Jump estimators for jump system," *IEEE Trans. on Automatic Control*, vol. 48, no. 22, pp. 2163–2171, Dec. 2003.

[74] J. Xu and J. P. Hespanha, "Estimation under uncontrolled and controlled communications in networked control systems," in *Proc. 43th IEEE Conf. on Decision and Control*, pp. 3527–3532, Dec. 2005.

[75] L. Ljung, *System Identification: Theory for the User*, Prentice Hall, Upper Saddle River, NJ, 1987.

[76] L. Zhang, B. Huang, and J. Lam, "H_∞ model reduction of Markovian jump linear systems," *Systems & Control Letters*, vol. 50, no. 2, pp. 103–118, Oct. 2003.

[77] L. E. Shaikhet, "Necessary and sufficient conditions of asymptotic mean-square stability for stochastic linear difference equations," *Applied Mathematics Letters*, vol. 10, no. 3, pp. 111–115, May 1997.

[78] S. Boyd, L. E. Chaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[79] A-max 26, maxon DC motor datasheet, [Online] Available: http://www.maxonmotorusa.com.

[80] Z.-H. Guan, C.-Y. Chen, G. Feng, and T. Li, "Optimal tracking performance limitation of networked control system with limited bandwidth and additive colored white Gaussian noise," *IEEE Trans. on Circuits and Systems–I: Regular Papers*, vol. 60, no. 1, pp. 189–198, Feb. 2013.

[81] Y.-C. Lin and F.-L. Lian, "Data reduction and bandwidth allocation for video-based network system," in *Proc. of IEEE International Conference on Information and Automation*, pp. 116–121, Jun. 2012.

[82] W. P. M. H. Heemels, A. R. Teel, N. V. D. Wouw, and D. Nešić, "Networked control systems with communication constraints: tradeoffs between transmission intervals, delays and performance," *IEEE Trans. on Automatic Control*, vol. 55, no. 8, pp. 1781–1796, Aug. 2010.

[83] L. Xu, M. Fei, T. Jia, and T. C. Yang, "Bandwidth scheduling and optimization using non-cooperative game model-based shuffled frog leaping algorithm in a networked learning control system," *Neural Comput. & Applic.*, vol. 21, no. 6, pp. 1117–1128, Sep. 2012.

[84] Q. Lin, Dynamic Scheduling of Real-Time Control Systems, M.S thesis, National University of Singapore, Singapore, 2004.

[85] C. Peng, D. Yue, Z. Gu, and F. Xia, "Sampling period scheduling of networked control systems with multiple-control loops," *Mathematics and Computer in Simulation*, vol. 79, no. 5, pp. 1502–1511, Jan. 2009.

[86] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K. Arzen, "How does control timing affect performance? Analysis and simulation of timing using Jitterbug and Truetime," *IEEE Control Systems Magazine*, vol. 23, no. 3, pp. 16–30, Jun. 2003.

[87] B. K. Kim, "Task scheduling with feedback latency for real-time control systems," in *Proc. of 5th Int. Conf. on Real-Time Computing Systems and Applications (RTCAS)*, pp. 37–41, Oct. 1998.

[88] I. Griva, S. G. Nash, and A. Sofer, *Linear and Nonlinear Optimization*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2008.

[89] M. Park, "Non-preemptive fixed priority scheduling of hard real-time periodic tasks," in *Proc. 7th Int. Conf. on Computational Science*, pp. 882–888, May 2007.

APPENDIX A

C/C++ CODES FOR THE NCS

A.1. C CODE FOR SERVER

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <asm/errno.h>
#include <sys/types.h>
#include <sys/user.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sched.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <errno.h>
#include <inttypes.h>
#include "defines.h"
#define KEEP_STATIC_INLINE
#include <rtai_lxrt_user.h>
#include <rtai_lxrt.h>
```

```
RTIME time_stamp;
double u0, u1e, u2e, u3e, u4e, u5e, u6e, u7e, u8e;
double y0, y_1, y_2, y_3, y_4, y_5, y_6, y_7;
char y_8[6]="00000";
double u_1, u_2;
double y_hat_1, y_hat_2, y_hat_3, y_hat_4, y_hat_5, y_hat_6, y_hat_7, y_hat_8;
doube delay;
FILE *fp;
// DC-motor (m5)
double y_dot_desi = 10.0;
double e0 = 0, e1 = 0, u0 = 0, u1 = 0, u2 = 0;
// A Ball Maglev
double y_hat_desi = 0.005;    //* User input (desired set point)
double v_hat_err = 0.0;
double k = 0.083;
double c = 0.0;               //* Controller constant
double v = 0.975;             //current setting
double er0 = 0.0, er1 = 0.0, er2 = 0.0;
int i=0, p1 = 0, p2 = 0, p3 = 0, p4 = 0;
//rtai declarations
unsigned long mtsk_name;
RT_TASK *mtsk;
struct sched_param mysched;
void terminate_normally(int signo)
{
        fflush(stdin);
        if(signo==SIGINT || signo==SIGTERM)
        {
                printf("Terminating the program normally\n");
```

```
                rt_make_soft_real_time();
                printf("MASTER TASK YIELDS ITSELF\n");
                rt_task_yield();
                printf("MASTER TASK STOPS THE PERIODIC TIMER\n");
                stop_rt_timer();
                printf("MASTER TASK DELETES ITSELF\n");
                rt_task_delete(mtsk);
                printf("END MASTER TASK\n");
        }
        exit(0);
}
main(int argc, char *argv[])
{
        int sockid, nread, addrlen;
        struct sockaddr_in my_addr, client_addr;
        int nw, nr;
        int send_buffer_size, recv_buffer_size;
        unsigned short server_port = 0;
        struct send_data *send_buffer = NULL;
        struct recv_data *recv_buffer = NULL;
        RTIME start_time = 0, end_time = 0, actual_period = 0, difference = 0;
        size_t iRet = 0;
        int esti_count = 0;
        double vhaterr_prev[5] = {0.0, 0.0, 0.0, 0.0, 0.0};
        int j=0;
        double h2=0.005, h3=0.05;
        float hh=1.0;
        double delay2=0, delay3=0;
        struct sigaction sa;
```

```
        sa.sa_handler = terminate_normally;
        sa.sa_flags = 0;
        sigemptyset(&sa.sa_mask);
        if(sigaction(SIGINT, &sa, NULL)) {
                perror("sigaction");
        }
        if(sigaction(SIGTERM, &sa, NULL)) {
                perror("sigaction");
        }
         fprintf(stderr, "creating socket\n");
         if ( (sockid = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
                perror("socket() failed ");
                fprintf(stderr, "%s: socket error: %d\n", argv[0], errno);
                exit(2);
        }
        fprintf(stderr, "binding my local socket\n");
         server_port = 4444;
        memset((void *) &my_addr, (char) 0, sizeof(my_addr));
        my_addr.sin_family = AF_INET;
        my_addr.sin_addr.s_addr = htons(INADDR_ANY);
        my_addr.sin_port = htons(server_port);
         if ( (bind(sockid, (struct sockaddr *) &my_addr, sizeof(my_addr)) < 0) ) {
        perror("bind() failed ");
        fprintf(stderr, "bind() errno = %d\n", errno);
         exit(4);
        }
        recv_buffer_size = sizeof(struct recv_data);
         if((recv_buffer = (struct recv_data *)calloc(1, sizeof(struct recv_data)))
==NULL) {
```

```
                fprintf(stderr, "cannot allocate memory for buffer!\n");
                exit(4);
        }
         send_buffer_size = sizeof(struct send_data);
         if((send_buffer = (struct send_data *)calloc(1, sizeof(struct send_data)))
==NULL) {
                fprintf(stderr, "cannot allocate memory for buffer!\n");
                exit(4);
        }
        addrlen = sizeof(client_addr);
        fprintf(stderr, "%s: starting blocking message read\n", argv[0]);
        mysched.sched_priority = 99;
        if( sched_setscheduler( 0, SCHED_FIFO, &mysched ) == -1 ) {
        puts(" ERROR IN SETTING THE SCHEDULER UP");
        perror( "errno" );
        exit( 0 );
        }
        mlockall(MCL_CURRENT | MCL_FUTURE);
        mtsk_name = nam2num("MTSK");
        if (!(mtsk = rt_task_init(mtsk_name, 0, 0, 0))) {
                printf("CANNOT INIT MASTER TASK\n");
                exit(1);
        }
        start_time = rt_get_cpu_time_ns();
        printf("main: start_time = %lld\n", start_time);
        printf("MASTER TASK STARTS THE ONESHOT TIMER\n");
        actual_period = start_rt_timer(nano2count(25000));
        printf("actual_period = %lld\n", actual_period);
        printf("MASTER TASK MAKES ITSELF PERIODIC \n");
```

```
        rt_task_make_periodic(mtsk, rt_get_time()+ nano2count(1000000),
nano2count(1000000));
        while( 1 )
        {
                nr = recvfrom(sockid, (void *)recv_buffer, recv_buffer_size, 0, (struct
sockaddr *) &client_addr, &addrlen);
                if( nr <= -1 ) {
                        fprintf(stderr, "recvfrom() errno = %d\n", errno);
                        exit(10);
                }
                start_time = rt_get_cpu_time_ns();
                y0 = recv_buffer->y_0;
                y_1 = recv_buffer->y_1;
                y_2 = recv_buffer->y_2;
                y_3 = recv_buffer->y_3;
                y_4 = recv_buffer->y_4;
                y_5 = recv_buffer->y_5;
                y_6 = recv_buffer->y_6;
                y_7 = recv_buffer->y_7;
                y_8[0] = recv_buffer->y_8[0];
                y_8[1] = recv_buffer->y_8[1];
                y_8[2] = recv_buffer->y_8[2];
                y_8[3] = recv_buffer->y_8[3];
                y_8[4] = recv_buffer->y_8[4];
                u_1 = recv_buffer->u_1;
                u_2 = recv_buffer->u_2;
                delay = recv_buffer->delay;
                if (delay2/h2 + delay3/h3 > 0.545) {
                        h2 = h2*1.05;
```

```
                h3 = h3*1.05;
        }
        if(y_7 == 1) {
                er0 = (y_hat_desi - (-0.0010108*y0+0.0114970))*k;
                er1 = (y_hat_desi - (-0.0010108*y_1+0.0114970))*k;
                er2 = (y_hat_desi - (-0.0010108*y_2+0.0114970))*k;
                u0 = ((0.782*(u_1-v)) + (0.13*(u_2-v)) - (41500.0*er0) +
(41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
                send_buffer->u0 = hh*u0;
                y_hat_1 = 0.8122*y0 - 0.3479*y_1 - 0.0294*y_2 + 0.4605*y_3 +
0.0742*y_4 + 0.1042*y_5 + 0.1117*y_6;// - 0.3561*y_7;
                er0 = (y_hat_desi - (-0.0010108*y_hat_1+0.0114970))*k;
                er1 = (y_hat_desi - (-0.0010108*y0+0.0114970))*k;
                er2 = (y_hat_desi - (-0.0010108*y_1+0.0114970))*k;
                u1e = ((0.782*(u0-v)) + (0.13*(u_1-v)) - (41500.0*er0) +
(41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
                send_buffer->u1e = hh*u1e;
                y_hat_2 = 0.3117*y0 - 0.3119*y_1 + 0.4366*y_2 + 0.4482*y_3 +
0.1645*y_4 + 0.1964*y_5 - 0.2653*y_6;// - 0.2892*y_7;
                er0 = (y_hat_desi - (-0.0010108*y_hat_2+0.0114970))*k;
                er1 = (y_hat_desi - (-0.0010108*y_hat_1+0.0114970))*k;
                er2 = (y_hat_desi - (-0.0010108*y0+0.0114970))*k;
                u2e = ((0.782*(u1e-v)) + (0.13*(u0-v)) - (41500.0*er0) +
(41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
                send_buffer->u2e = hh*u2e;
                y_hat_3 = -0.0587*y0 + 0.3281*y_1 + 0.4390*y_2 + 0.3080*y_3
+ 0.2195*y_4 - 0.2329*y_5 - 0.2544*y_6;// - 0.1110*y_7;
                er0 = (y_hat_desi - (-0.0010108*y_hat_3+0.0114970))*k;
                er1 = (y_hat_desi - (-0.0010108*y_hat_2+0.0114970))*k;
```

```
er2 = (y_hat_desi - (-0.0010108*y_hat_1+0.0114970))*k;
u3e = ((0.782*(u2e-v)) + (0.13*(u1e-v)) - (41500.0*er0) +
(41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
send_buffer->u3e = hh*u3e;
y_hat_4 = 0.2804*y0 + 0.4594*y_1 + 0.3097*y_2 + 0.1925*y_3 -
0.2372*y_4 - 0.2605*y_5 - 0.1176*y_6;// + 0.0209*y_7;
er0 = (y_hat_desi - (-0.0010108*y_hat_4+0.0114970))*k;
er1 = (y_hat_desi - (-0.0010108*y_hat_3+0.0114970))*k;
er2 = (y_hat_desi - (-0.0010108*y_hat_2+0.0114970))*k;
u4e = ((0.782*(u3e-v)) + (0.13*(u2e-v)) - (41500.0*er0) +
(41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
send_buffer->u4e = hh*u4e;
y_hat_5 = 0.6872*y0 + 0.2122*y_1 + 0.1842*y_2 - 0.1081*y_3 -
0.2397*y_4 - 0.0884*y_5 + 0.0523*y_6;// - 0.0999*y_7;
er0 = (y_hat_desi - (-0.0010108*y_hat_5+0.0114970))*k;
er1 = (y_hat_desi - (-0.0010108*y_hat_4+0.0114970))*k;
er2 = (y_hat_desi - (-0.0010108*y_hat_3+0.0114970))*k;
u5e = ((0.782*(u4e-v)) + (0.13*(u3e-v)) - (41500.0*er0) +
(41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
send_buffer->u5e = hh*u5e;
y_hat_6 = 0.7703*y0 - 0.0548*y_1 - 0.1283*y_2 + 0.0767*y_3 -
0.0374*y_4 + 0.1239*y_5 - 0.0231*y_6;// - 0.2447*y_7;
er0 = (y_hat_desi - (-0.0010108*y_hat_6+0.0114970))*k;
er1 = (y_hat_desi - (-0.0010108*y_hat_5+0.0114970))*k;
er2 = (y_hat_desi - (-0.0010108*y_hat_4+0.0114970))*k;
u6e = ((0.782*(u5e-v)) + (0.13*(u4e-v)) - (41500.0*er0) +
(41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
send_buffer->u6e = hh*u6e;
```

```
                    y_hat_7 = 0.5708*y0 - 0.3963*y_1 + 0.0541*y_2 + 0.3173*y_3 +
0.1810*y_4 + 0.0572*y_5 - 0.1586*y_6;// - 0.2743*y_7;
                    er0 = (y_hat_desi - (-0.0010108*y_hat_7+0.0114970))*k;
                    er1 = (y_hat_desi - (-0.0010108*y_hat_6+0.0114970))*k;
                    er2 = (y_hat_desi - (-0.0010108*y_hat_5+0.0114970))*k;
                    u7e = ((0.782*(u6e-v)) + (0.13*(u5e-v)) - (41500.0*er0) +
(41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
                    send_buffer->u7e = hh*u7e;
                    send_buffer->u8e = 1;
                    p1 = p1+1;
                    printf("m02  %d\n",p1);
               }
               if(y_7 == 2) {
                    delay2 = delay;
                    e1 = y_dot_desi - y0;
                    if (delay2 <=0.00046)
                         u0 = 1.0102*u1 +1.7621 *e1 - 1.7417*e0;
                    else if (delay2 <=0.00063)
                         u0 = 1.0155*u1 + 1.7889*e1  - 1.7762 *e0;
                    else if (delay2 <=0.0008)
                         u0 = 1.0412*u1 +1.8162*e1  - 1.8488*e0;
                    else
                         u0 = 1.1974*u1 + 2.0866*e1 – 2.4366*e0;
                    send_buffer->u0 = u0;
                    u1e= 0.5094*u0 + 0.4094*u2;
                    send_buffer->u1e = u1e;
                    u2e = 0.6136*u1 + 0.3862*u2;
                    send_buffer->u2e = u2e;
                    u3e = 0;
```

```
            send_buffer->u3e = u3e;
            u4e = 0;
            send_buffer->u4e = u4e;
            u5e = 0;
            send_buffer->u5e = u5e;
            u6e = 0;
            send_buffer->u6e = u6e;
            u7e = 0;
            send_buffer->u7e = u7e;
            u8e = 2;
            send_buffer->u8e = u8e;
            send_buffer->h = h2;
            p2 = p2+1;
            printf("            m05 %d\n", p2);
            u1 = u0;
            u2 = u1;
            e0 = e1;
    }
if(y_7 == 3) {
            delay3 = delay;
            if(strcmp(y_8,"00010")==0){
                        send_buffer->u0 = 2.0;
            }
            else if(strcmp(y_8,"00011")==0){
                        send_buffer->u0 = 2.0;
            }
            else if(strcmp(y_8,"00100")==0){
                        send_buffer->u0 = 2.0;
            }
```

```
else if(strcmp(y_8,"00101")==0){
        send_buffer->u0 = 2.0;
}
else if(strcmp(y_8,"00110")==0){
        send_buffer->u0 = 2.0;
}
else if(strcmp(y_8,"00111")==0){
        send_buffer->u0 = 2.0;
}
else if(strcmp(y_8,"01000")==0){
        send_buffer->u0 = 7.0;
}
else if(strcmp(y_8,"01001")==0){
        send_buffer->u0 = 7.0;
}
else if(strcmp(y_8,"01010")==0){
        send_buffer->u0 = 10.0;
}
else if(strcmp(y_8,"01100")==0){
        send_buffer->u0 = 10.0;
}
else if(strcmp(y_8,"01110")==0){
        send_buffer->u0 = 2.0;
}
else if(strcmp(y_8,"10000")==0){
        send_buffer->u0 = 7.0;
}
else if(strcmp(y_8,"10001")==0){
        send_buffer->u0 = 7.0;
```

```
}
else if(strcmp(y_8,"10010")==0){
        send_buffer->u0 = 10.0;
}
else if(strcmp(y_8,"10100")==0){
        send_buffer->u0 = 10.0;
}
else if(strcmp(y_8,"10110")==0){
        send_buffer->u0 = 10.0;
}
else if(strcmp(y_8,"11000")==0){
        send_buffer->u0 = 7.0;
}
else if(strcmp(y_8,"11001")==0){
        send_buffer->u0 = 7.0;
}
else if(strcmp(y_8,"11010")==0){
        send_buffer->u0 = 10.0;
}
else if(strcmp(y_8,"11100")==0){
        send_buffer->u0 = 7.0;
}
else if(strcmp(y_8,"11110")==0){
        send_buffer->u0 = 10.0;
}
else if(strcmp(y_8,"00000")==0){
        send_buffer->u0 = 10.0;
}
else{
```

```
                send_buffer->u0 = 5.0;
            }
            u1e = 0;
            send_buffer->u1e = u1e;
            u2e = 0;
            send_buffer->u2e = u2e;
            u3e = 0;
            send_buffer->u3e = u3e;
            u4e = 0;
            send_buffer->u4e = u4e;
            u5e = 0;
            send_buffer->u5e = u5e;
            u6e = 0;
            send_buffer->u6e = u6e;
            u7e = 0;
            send_buffer->u7e = u7e;
            u8e = 2;
            send_buffer->u8e = u8e;
            send_buffer->h = h3;
            p3 = p3+1;
            printf("                              _wheelchair %d_\n",p3);
            u1 = u0;
            e0 = e1;
         }
        end_time = rt_get_cpu_time_ns();
        send_buffer->time_stamp = recv_buffer->time_stamp;
         nw = sendto(sockid, (const void *)send_buffer, send_buffer_size, 0,
(struct sockaddr *) &client_addr, addrlen);
        if( nw <= -1 ) {
```

```
                    perror("sendto failed ");
                    fprintf(stderr, "sendto() errno = %d \n", errno);
                    exit(12);
                }
        } // END of while
        fclose(fp);
        printf("MASTER TASK YIELDS ITSELF\n");
        rt_task_yield();
        printf("MASTER TASK STOPS THE PERIODIC TIMER\n");
        stop_rt_timer();
        printf("MASTER TASK DELETES ITSELF\n");
        rt_task_delete(mtsk);
        close(sockid);
        free(send_buffer);
        free(recv_buffer);
}
```

A.2. C CODE FOR CLIENT (DC MOTORS)

```
#include <stdio.h>
…       //* here is an identical code block as the one in Appendix A.1
#define PERIOD 1000000
#define LOOPS 20000
#define NTASKS 2
#define taskname(x) (1000 + (x))
…       //* here is an identical code block as the one in Appendix A.1
double y_7=2;
double u_1, u_2;
float h = 3;
RTIME current_time_stamp;
int j = 0, m = 0, b0 = 0, b1 = 0, cnt = 0, p = 0;
```

```
double speed = 0;
pthread_t task[NTASKS];
int ntasks = NTASKS;
RT_TASK *mytask;
SEM *sem;     //added static
static int cpus_allowed;
SEM *sock_sem;       //socket semaphoere, used by all the threads.
int sockid;
RTIME start_instant;
int server_sock_size = 0;
struct sockaddr_in my_addr, server_addr;
comedi_t *it;
int in_subdev = 2;       //digital input
int out_subdev = 1;     //analog output
int in_chan = 0, out_chan = 0, in_range = 0, out_range = 0;
int aref = AREF_GROUND;
int i=0;
//comedi declarations
lsampl_t in_data;
lsampl_t out_data;
float volts = 0.0;
int in_maxdata = 0, out_maxdata = 0;
comedi_range *in_range_ptr, *out_range_ptr;
int endme_int = 0;
void terminate_normally(int signo);
void endme(int sig)
{
        printf("You want to kill me?\n");
        endme_int = 1;
```

```
        rt_sem_delete(sem);
        comedi_close(it);
        stop_rt_timer();
        rt_task_delete(mytask);
        signal(SIGINT, SIG_DFL);
        exit(1);
}
void *send_thread_fun(void *arg)
{
        RTIME start_time, period, end_time, difference;
        RTIME t0;
        SEM *sem;
        RT_TASK *mytask;
        unsigned long mytask_name;
        int mytask_indx;
        int iRet = 0;
        struct recv_data *send_msg = NULL;
        int send_msg_size;
        FILE *fp = NULL;
        float print_data[LOOPS];
        int loop_count = 0;
        fp = fopen("result.txt","w");
        if(fp == NULL)
        {
                printf("could not open file");
                exit(0);
        }
        pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
        mytask_indx = 0;
```

```
	mytask_name = taskname(mytask_indx);
	cpus_allowed = 1 - cpus_allowed;
	if (!(mytask = rt_task_init_schmod(mytask_name, 1, 0, 0, SCHED_FIFO, 1 <<
cpus_allowed))) {
		printf("CANNOT INIT send_thread TASK\n");
		exit(1);
	}
	printf("send thread pid = %d\t master pid = %d\n", getpid(), getppid());
	mlockall(MCL_CURRENT | MCL_FUTURE);
	rt_receive(0, (unsigned int*)&sem);
	send_msg_size = sizeof(struct recv_data);
	if(( send_msg = (struct recv_data *)calloc(1, sizeof(struct recv_data))) ==
NULL){
	printf("cannot allocate message memory\n");
		exit(4);
	}
	period = nano2count(PERIOD);
	start_time = rt_get_time() + nano2count(10000000);
	t0 = start_instant;
	printf("send: t0 = %lld\t", t0);
	printf("This period = %lld\t", rt_get_time());
	printf("actual start = %lld\n", t0 + nano2count(500000000));
	rt_task_make_periodic(mytask, (t0 + nano2count(500000000)),
nano2count(h*1000000));
	printf("starting the send_thread while loop\n");
	// DC motor speed measure
	while(1){
		if(endme_int == 1){
			break;
```

```
                }
                // Counting encoder pulses
                start_time = rt_get_cpu_time_ns();
                comedi_dio_config(it, in_subdev, in_chan, COMEDI_INPUT);
                for(j=0;j<500;j++) {
                        m = comedi_data_read(it, in_subdev, in_chan, in_range, aref,
&in_data);
                        if(in_data == 1)        //* high or low?
                                {b1 = 1;}
                        else
                                {b1 = 0;}
                        if(b1 != b0)    //* count turn-over (H to L or L to H)
                        {cnt++;}
                        b0 = b1;
                }
                end_time = rt_get_cpu_time_ns(); //* End of the FOR loop
                current_time_stamp = rt_get_cpu_time_ns();
                speed = (cnt*0.214)+1.322;    //* DI counting w/ j=1000, PCI-6025E
                y_0 = speed;
                send_msg->y_0 = y_0;
                send_msg->y_1 = y_1;
                send_msg->y_2 = y_2;
                send_msg->y_3 = y_3;
                send_msg->y_4 = y_4;
                send_msg->y_5 = y_5;
                send_msg->y_6 = y_6;
                send_msg->y_7 = y_7;
                send_msg->u_1 = u_1;
                send_msg->u_2 = u_2;
```

```
                send_msg->time_stamp = current_time_stamp;
                send_msg->delay = time_diff[loop_count]
                rt_sem_wait(sock_sem);
                iRet = sendto(sockid, (const void *)send_msg, send_msg_size, 0, (struct
sockaddr*)&server_addr, server_sock_size);
                rt_sem_signal(sock_sem);
                if(iRet <= -1) {
                        perror("sendto() failed\n");
                        break;
                }
                //y_7 = y_6;
                y_6 = y_5;
                y_5 = y_4;
                y_4 = y_3;
                y_3 = 0.7151*y_2+0.2848*y_1;
                y_2 = 0.6867*y_1+0.3182*y_0;
                y_1 = y_0;
                cnt = 0;
                loop_count++;
                if(loop_count == LOOPS) {
                        break;
                }
                rt_task_wait_period();
                print_data[loop_count] = y_0;
        }
        end_time = rt_get_cpu_time_ns();
        difference = end_time - start_time;
        printf("difference = %lld\n", difference);
        endme_int++;
```

```
        rt_sem_signal(sem);
        rt_make_soft_real_time();
        for(i=0;i<LOOPS;i++) {
                fprintf(fp, "%f\n", print_data[i]);
        }
        fclose(fp);
        free(send_msg);
        rt_task_delete(mytask);
        printf("send_thread ENDS\n");
        return 0;
}
void *recv_thread_fun(void *arg)
{
        RTIME start_time, period, end_time, difference = 0;
        RTIME t0;
        RTIME re_time_stamp, current_cpu_time;
        SEM *sem;
        RT_TASK *mytask;
        unsigned long mytask_name;
        int mytask_indx;
        int iRet = 0;
        int recv_msg_size;
        struct send_data *recv_msg = NULL;
        int loop_count = 0;
        float control_data[LOOPS];
        RTIME time_diff[LOOPS];
        FILE *tdiff = NULL;
        FILE *control = NULL;
        tdiff = fopen("timediff.txt","w");
```

```
        if(tdiff == NULL){
                printf("could not open tdiff file");
                exit(0);
        }
        control = fopen("control.txt","w");
        if(control == NULL) {
                printf("could not open control file");
                exit(0);
        }
        recv_msg_size = sizeof(struct send_data);
        if(( recv_msg = (struct send_data *)calloc(1, sizeof(struct send_data))) ==
NULL) {
                printf("cannot allocate message memory\n");
                exit(4);
        }
        pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
        mytask_indx = 1;
        mytask_name = taskname(mytask_indx);
        cpus_allowed = 1 - cpus_allowed;
        if (!(mytask = rt_task_init_schmod(mytask_name, 1, 0, 0, SCHED_FIFO, 1 <<
cpus_allowed))) {
                printf("CANNOT INIT recv_thread TASK\n");
                exit(1);
        }
        printf("recv thread pid = %d\t master pid = %d\n", getpid(), getppid());
        mlockall(MCL_CURRENT | MCL_FUTURE);
        rt_receive(0, (unsigned int*)&sem);
        period = nano2count(PERIOD);
        start_time = rt_get_time() + nano2count(10000000);
```

```
	t0 = start_instant;
	printf("recv: t0 = %lld\t", count2nano(t0));
	printf("This period = %lld\t", count2nano(rt_get_time()));
	printf("actual start = %lld\n", count2nano(t0 + nano2count(500500000)));
	rt_task_make_periodic(mytask, (t0 + nano2count(500500000)),
nano2count(h*1000000));
	start_time = rt_get_time();
	printf("starting the recv_thread while loop\n");
	for(;;) {
		if(endme_int == 1) {
			break;
		}
		rt_sem_wait(sock_sem);
		iRet = recvfrom(sockid, (void *)recv_msg, recv_msg_size, 0, (struct
sockaddr *)&server_addr, &server_sock_size);
		rt_sem_signal(sock_sem);
		if(iRet <= -1) {
			endme_int = 1;
			printf("difference = %lld\n", difference);
			perror("recvfrom() failed\n");
			break;
		}
		if(loop_count < LOOPS) {
		u0 = recv_msg->u0;
		u1e = recv_msg->u1e;
		u2e = recv_msg->u2e;
		u3e = recv_msg->u3e;
		u4e = recv_msg->u4e;
		u5e = recv_msg->u5e;
```

```
u6e = recv_msg->u6e;
u7e = recv_msg->u7e;
u8e = recv_msg->u8e;
h = recv_msg->h;
re_time_stamp = recv_msg->time_stamp;
current_cpu_time = rt_get_cpu_time_ns();
time_diff[loop_count] = current_cpu_time-re_time_stamp;
control_data[loop_count] = u0;
}
if(loop_count < (LOOPS-2)){
        volts = u0*1.014-0.005;
}
else if(loop_count == (LOOPS-2)) {
        volts = 0.0;
}
if(volts > 5.0)  {
        volts = 4.99999;
}
if(volts < 0.0) {
        volts = 0.0;
}
out_data = comedi_from_phys(volts, out_range_ptr, out_maxdata);
comedi_data_write(it, out_subdev, out_chan, out_range, aref, out_data);
u_1 = u0;
if(loop_count == LOOPS) {
        loop_count = 0;
}
else {
        loop_count++;
```

```
                }
                rt_task_wait_period();
        }
        end_time = rt_get_cpu_time_ns();
        difference = end_time - start_time;
        printf("difference = %lld\n", difference);
        endme_int++;
        rt_make_soft_real_time();
        for(i=0;i<LOOPS;i++) {
                fprintf(tdiff, "%lld\n", time_diff[i]);
                fprintf(control, "%f\n", control_data[i]);
        }
        fclose(tdiff);
        fclose(control);
        free(recv_msg);
        rt_task_delete(mytask);
        printf("recv_thread ENDS\n");
        return 0;
}//End of  Recv Thread
int main(void)
{
        int i;
        unsigned long mytask_name = nam2num("MASTER");
        struct sigaction sa;
        char * server_ip = "165.91.95.40"; //maglev1
         unsigned short my_port, server_port;
        my_port = 4445;
        server_port = 4444;
        printf("creating socket\n");
```

```
if( (sockid = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket() failed ");
        exit(2);
}
memset((void *) &my_addr, (char) 0, sizeof(my_addr));
my_addr.sin_family = AF_INET;
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
my_addr.sin_port = htons(my_port);
if ( (bind(sockid, (struct sockaddr *) &my_addr, sizeof(my_addr)) < 0) ) {
        perror("bind() failed ");
        exit(3);
}
server_sock_size = sizeof(server_addr);
memset((void *) &server_addr, (char) 0, server_sock_size);
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = inet_addr(server_ip);
server_addr.sin_port = htons(server_port);
sa.sa_handler = endme;
sa.sa_flags = 0;
sigemptyset(&sa.sa_mask);
if(sigaction(SIGINT, &sa, NULL)) {
        perror("sigaction");
}
if(sigaction(SIGTERM, &sa, NULL)) {
        perror("sigaction");
}
it = comedi_open("/dev/comedi0");
if(it == NULL) {
        printf("Could not open comedi\n");
```

```
        exit(1);
}
in_maxdata = comedi_get_maxdata(it, in_subdev, in_chan);
out_maxdata = comedi_get_maxdata(it, out_subdev, out_chan);
in_range_ptr = comedi_get_range(it, in_subdev, in_chan, in_range);
out_range_ptr = comedi_get_range(it, out_subdev, out_chan, out_range);
if (!(mytask = rt_task_init(mytask_name, 1, 0, 0))) {
        printf("CANNOT INIT main TASK \n");
        exit(1);
}
printf("MASTER INIT: name = %lu, address = %p.\n", mytask_name, mytask);
sem = rt_sem_init(10000, 0);
sock_sem = rt_sem_init(nam2num("SOCK"), 1);
rt_set_periodic_mode();
start_rt_timer(nano2count(25000));
start_instant = rt_get_time();
printf("main: start_instant = %lld\n", start_instant);
if (pthread_create(&task[0], NULL, send_thread_fun, &start_instant)) {
        printf("ERROR IN CREATING send_thread\n");
        exit(1);
}
if (pthread_create(&task[1], NULL, recv_thread_fun, &start_instant)) {
        printf("ERROR IN CREATING recv_thread\n");
        exit(1);
}
for (i = 0; i < ntasks; i++) {
        while (!rt_get_adr(taskname(i))) {
                rt_sleep(nano2count(20000000));
        }
```

```
        }
        for (i = 0; i < ntasks; i++) {
                rt_send(rt_get_adr(taskname(i)), (unsigned int)sem);
        }
        printf("Start waiting for sem\n");
        while(endme_int == 0) {
                rt_sem_wait_timed(sem, nano2count(50000000));
        }
        printf("Stop waiting for sem\n");
        for (i = 0; i < ntasks; i++) {
                while (rt_get_adr(taskname(i))) {
                        rt_sleep(nano2count(20000000));
                }
        }
        rt_sem_delete(sem);
        rt_sem_delete(sock_sem);
        stop_rt_timer();
        comedi_close(it);
        rt_task_delete(mytask);
        printf("MASTER %lu %p ENDS\n", mytask_name, mytask);
        for (i = 0; i < ntasks; i++) {
                pthread_join(task[i], NULL);
        }
        return 0;
}
```

A.3. C CODE FOR INTEROPERABILITY SUITE

```
#include <stdio.h>
…       //*here is an identical code block as the one in Appendix A.1
double u0, u1e, u2e, u3e, u4e, u5e, u6e, u7e, u8e;
```

```
char y_8[6]="000000";
double y_0, y_1, y_2, y_3, y_4, y_5, y_6;
double y_7=3;
double u_1, u_2;
double delay, h;
//rtai declarations
…       //* here is an identical code block as the one in Appendix A.1
void terminate_normally(int signo)
{
…       //* here is an identical code block as the one in Appendix A.1
}
int main(int argc, char *argv[])
{
        …       //* here is an identical code block as the one in Appendix A.1
        char recv_msg[6] = "000000"; //Client to Interoperability Suite
        char fwd[10] = "front";
        char back[10] = "back";
        char stop[10] = "stop";
        char left[10] = "left";
        char right[10] = "right";
        char *server_ip= "165.91.95.40";
        FILE *fp = NULL;
        fp = fopen("result.txt","w");
        if (fp==NULL) {
                printf("could not open file\n");
                exit(0);
        }
        RTIME start_time = 0, end_time = 0, actual_period = 0;
        …       //* here is an identical code block as the one in Appendix A.1
```

```
        fprintf(stderr, "binding sockets\n");
        server_port = 4444;
        second_port = 3333;
        addrlen = sizeof(server_addr);
        clilen = sizeof(my_addr);
        memset((void *) &server_addr, (char) 0, addrlen);
        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = inet_addr(server_ip);
        server_addr.sin_port = htons(server_port);
        memset((void *) &my_addr, (char) 0, clilen);
        my_addr.sin_family = AF_INET;
        my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        my_addr.sin_port = htons(second_port);
        if ( (bind(sd, (struct sockaddr *) &my_addr,sizeof(my_addr)) < 0) ){
                perror("2bind() failed ");
                fprintf(stderr, "bind() errno = %d\n", errno);
                exit(4);
        }
        recv_buffer_size = sizeof(struct recv_data);
        if(( recv_buffer = (struct recv_data *)calloc(1, sizeof(struct recv_data)))
==NULL){
        fprintf(stderr, "cannot allocate memory for buffer!\n");
                exit(4);
        }
        send_buffer_size = sizeof(struct send_data);
        if(( send_buffer = (struct send_data *)calloc(1, sizeof(struct send_data)))
==NULL){
        fprintf(stderr, "cannot allocate memory for buffer!\n");
                exit(4);
```

```
        }
        fprintf(stderr, "%s: starting blocking message read\n", argv[0]);
        …       //* here is an identical code block as the one in Appendix A.1
        start_time = rt_get_cpu_time_ns();
        printf("main: start_time = %lld\n", start_time);
        printf("MASTER TASK STARTS THE ONESHOT TIMER\n");
        //rt_set_oneshot_mode();
        actual_period = start_rt_timer(nano2count(25000));
        printf("actual_period = %lld\n", actual_period);
        printf("MASTER TASK MAKES ITSELF PERIODIC \n");
        rt_task_make_periodic(mtsk, rt_get_time()+ nano2count(3000000),
nano2count(3000000));
        while( 1 ) {
                cr = recvfrom(sd, recv_msg, 10, 0, (struct sockaddr *) &client_addr,
&clilen);
                if( cr <= -1 ){
                        fprintf(stderr, "2recvfrom() errno = %d\n", errno);
                        exit(10);
                }
                start_time = rt_get_cpu_time_ns();
                y_1 = 0;
                y_2 = 0;
                y_3 = 0;
                y_4 = 0;
                y_5 = 0;
                y_6 = 0;
                y_7 = 3;
                y_8[0] = recv_msg[0];
                y_8[1] = recv_msg[1];
```

```
y_8[2] = recv_msg[2];
y_8[3] = recv_msg[3];
y_8[4] = recv_msg[4];
y_8[5] = recv_msg[5];
u_1 = 0;
u_2 = 0;
…        //* here is an identical code block as the one in Appendix A.2
send_buffer->time_stamp = current_time_stamp;
send_buffer->delay = delay;
nw=sendto(sockid, (const void *)send_buffer, send_buffer_size, 0,(struct
sockaddr *) &server_addr, addrlen);
if( nw <= -1 ){
        perror("1sendto failed ");
        fprintf(stderr, "sendto() errno = %d \n", errno);
        exit(12);
}
nr = recvfrom(sockid, (void *)recv_buffer, recv_buffer_size, 0, (struct
sockaddr *) &server_addr, &addrlen);
if( nr <= -1 ){
        fprintf(stderr, "1recvfrom() errno = %d\n", errno);
        exit(10);
}
…        //* here is an identical code block as the one in Appendix A.1
printf("recv: %s   u0: %f" ,y_8, u0);
if(u0 == 10.0){
        cw = sendto(sd, fwd, 6, 0, (struct sockaddr *) &client_addr,
clilen);
}
else if(u0 == 7.0){
```

```
                cw = sendto(sd, right, 6, 0, (struct sockaddr *) &client_addr,
clilen);
            }
            else if(u0 == 2.0){
                cw = sendto(sd, left, 6, 0, (struct sockaddr *) &client_addr,
clilen);
            }
            else if(u0 == 0.0){
                cw = sendto(sd, back, 6, 0, (struct sockaddr *) &client_addr,
clilen);
            }
            else if(u0 == 5.0){
                cw = sendto(sd, stop, 6, 0, (struct sockaddr *) &client_addr,
clilen);
            }
            end_time = rt_get_cpu_time_ns();
            send_buffer->time_stamp = recv_buffer->time_stamp;
            printf("end_time - start_time = %lld\n", (end_time - start_time));
            cnt = cnt +1;
        } //end while
…       //* here is an identical code block as the one in Appendix A.1
}
```

A.4. C++ CODE FOR CLIENT (WHEELCHAIR)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <NIDAQmx.h>
#include <winsock2.h>
#include <ws2tcpip.h>
```

```
#include <windows.h>
#pragma comment(lib, "winmm.lib")
#define MAXLOOP 200
#define DAQmxErrChk(functionCall) if( DAQmxFailed(error=(functionCall)) ) goto
Error; else
void move(uInt8 direction[8]);
void sensors();
int LIFR, LIFRS, RIFR, RIFRS, IFR;
void main(void)
{
        WSADATA w;          /* Used to open Windows connection */
        SOCKET sd;          /* The socket descriptor */
        int server_length;      /* Length of server struct */
        struct sockaddr_in server;      /* Information about the server */
        struct sockaddr_in client;      /* Information about the client */
        char *server_ip = "165.91.95.119";
        unsigned short server_port = 3333;
        char recv_data[6]="wheel", send_data[6]="00000";
        uInt8 forward[8]={0,1,1,0,0,1,1,0};
        uInt8 backward[8]={1,0,1,0,1,0,1,0};
        uInt8 left[8]={0,1,1,0,1,0,1,0};
        uInt8 right[8]={1,0,1,0,0,1,1,0};
        uInt8 stop[8]={0,0,0,0,0,0,0,0};
        int counter=0;
        int32 error=0;
        TaskHandle    taskHandle=0;
        uInt8 data[8];
        char errBuff[2048]={'\0'};
        int32 read,bytesPerSamp;
```

```
unsigned long starttime, endtime, timediff;
int pos[2];
int left_cnt=0, right_cnt=0, ileft=0, iright=0, h = 80;
FILE *fp;
errno_t err;
if ((err = fopen_s(&fp,"result.txt","w"))!=0) {
        printf("Can not open file! \n");
        exit(0);
}
timeBeginPeriod(1);
if (WSAStartup(0x0101, &w) != 0) {
        printf("Could not open Windows connection.\n");
        exit(0);
}
sd = socket(AF_INET, SOCK_DGRAM, 0);
if (sd == INVALID_SOCKET) {
        printf("Could not create socket.\n");
        WSACleanup();
        exit(0);
}
memset((void *)&server, '\0', sizeof(struct sockaddr_in));
server.sin_family = AF_INET;
server.sin_port = htons(server_port);
server.sin_addr.S_un.S_addr = inet_addr(server_ip);
memset((void *)&client, '\0', sizeof(struct sockaddr_in));
client.sin_family = AF_INET;
client.sin_port = htons(0);
client.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
if (bind(sd, (struct sockaddr *)&client, sizeof(struct sockaddr_in)) == -1)
```

```
	{
		printf("Cannot bind address to socket.\n");
		closesocket(sd);
		WSACleanup();
		exit(0);
	}
	printf("Wheelchair is ready.\n");
	printf("Wheelchair is running.\n");
	DAQmxErrChk (DAQmxCreateTask("",&taskHandle));
	DAQmxErrChk
(DAQmxCreateDIChan(taskHandle,"Dev1/port0/line0:7","",DAQmx_Val_ChanForAllL
ines));
	DAQmxErrChk (DAQmxStartTask(taskHandle));
	while (counter<MAXLOOP)
	{
		Sleep(200);
		DAQmxErrChk
(DAQmxReadDigitalLines(taskHandle,1,10.0,DAQmx_Val_GroupByChannel,data,8,&r
ead,&bytesPerSamp,NULL));
		starttime = timeGetTime();
		left_cnt = 1 * data[0] + 2 * data[1] + 4 * data[2] + 8 * data[3];
		right_cnt = 1 * data[4] + 2 * data[5] + 4 * data[6] + 8 * data[7];
		if (left_cnt < 8)
			ileft++;
		if (right_cnt < 8)
			iright++;
		pos[0] = ileft * 15 + left_cnt;
		pos[1] = iright * 15 + right_cnt;
		sensors();
```

```
                send_data[0] = (char)(((int)'0')+LIFR);
                send_data[1] = (char)(((int)'0')+LIFRS);
                send_data[2] = (char)(((int)'0')+RIFR);
                send_data[3] = (char)(((int)'0')+RIFRS);
                send_data[4] = (char)(((int)'0')+IFR);
                server_length = sizeof(struct sockaddr_in);
                if (sendto(sd, (char *)&send_data, (int)strlen(send_data) + 1, 0, (struct
sockaddr *)&server, server_length) == -1) {
                        printf("Error transmitting data.\n");
                        closesocket(sd);
                        WSACleanup();
                        exit(0);
                }
                if (recvfrom(sd, (char *)&recv_data, (int)sizeof(recv_data), 0, (struct
sockaddr *)&server, &server_length) < 0) {
                        printf("Error receiving data.\n");
                        closesocket(sd);
                        WSACleanup();
                        exit(0);
                }
                if (strcmp(recv_data,"stop")==0)
                        move(stop);
                else if (strcmp(recv_data,"back")==0)
                        move(backward);
                else if (strcmp(recv_data,"left")==0)
                        move(left);
                else if (strcmp(recv_data,"right")==0)
                        move(right);
                else move(forward);
```

```
                endtime = timeGetTime();
                timediff = endtime - starttime;
                fprintf(fp,"%d, %d\n",pos[0],pos[1]);
                Sleep(h);
                counter++;
                printf("Current loop: %d.  Command from the server: %s \n", counter,
recv_data);
        }
        move(stop);
        printf("Wheelchair stops. \n");
        closesocket(sd);
        WSACleanup();
        printf("To quit and close the console window, press any key! \n");
        timeEndPeriod(1);
        getchar();
        fclose(fp);
Error:
        if( DAQmxFailed(error))
                DAQmxGetExtendedErrorInfo(errBuff,2048);
        if( taskHandle!=0 ) {
                DAQmxStopTask(taskHandle);
                DAQmxClearTask(taskHandle);
        }
        if( DAQmxFailed(error) )
                printf("DAQmx Error: %s\n",errBuff);
}
void sensors(void)
{
        int32           error=0;
```

```
	TaskHandle	taskHandle=0;
	uInt8		data[8];
	char		errBuff[2048]={'\0'};
	int32		read, bytesPerSamp;
	DAQmxErrChk (DAQmxCreateTask("",&taskHandle));
	DAQmxErrChk
(DAQmxCreateDIChan(taskHandle,"Dev1/port1/line0:7","",DAQmx_Val_ChanForAllL
ines));
	DAQmxErrChk (DAQmxStartTask(taskHandle));
	DAQmxErrChk
(DAQmxReadDigitalLines(taskHandle,1,10.0,DAQmx_Val_GroupByChannel,data,8,&r
ead,&bytesPerSamp,NULL));
	LIFR = data[0];
	LIFRS = data[1];
	RIFR = data[2];
	RIFRS = data[3];
	IFR = data[4];
	photocell[0] = data[5];
	photocell[1] = data[6];
	photocell[2] = data[7];
Error:
	if( DAQmxFailed(error) )
		DAQmxGetExtendedErrorInfo(errBuff,2048);
	if( taskHandle!=0 ) {
		DAQmxStopTask(taskHandle);
		DAQmxClearTask(taskHandle);
	}
	if( DAQmxFailed(error) )
		printf("DAQmx Error: %s\n",errBuff);
```

```
}
void move(uInt8 direction[8])
{
	double	error=0;
	TaskHandle  taskHandle=0;
	char	errBuff[2048]={'\0'};
	DAQmxErrChk (DAQmxCreateTask("",&taskHandle));
	DAQmxErrChk
(DAQmxCreateDOChan(taskHandle,"Dev1/port2/line0:7","",DAQmx_Val_ChanForAll
Lines));
	DAQmxErrChk (DAQmxStartTask(taskHandle));
	DAQmxErrChk
(DAQmxWriteDigitalLines(taskHandle,1,1,10.0,DAQmx_Val_GroupByChannel,directi
on,NULL,NULL));
Error:
	if( DAQmxFailed(error))
		DAQmxGetExtendedErrorInfo(errBuff,2048);
	if( taskHandle!=0 ) {
		DAQmxStopTask(taskHandle);
		DAQmxClearTask(taskHandle);
	}
	if( DAQmxFailed(error))
		printf("DAQmx Error: %s\n",errBuff);
}
```

APPENDIX B

MATLAB® CODES AND SIMULINK® BLOCK DIAGRAMS

B.1. MATLAB® CODES FOR THE OUTPUT FEEDBACK CONTROLLER

```
A = -0.26; B = 2.04; C = 1;
Q = 1; R = 1;
[K, S, e] = LQR(A, B, Q, R);
K0 = K; K1 = K;
 P = [0.97 0.07; 0.75 0.25];
P0 = [0.969 0.068; 0.73 0.26];
delta = [0.001 -0.002; -0.02 0.01];
AA = [A 0; 1 0]; BB = [B; 0]; CC0 = [0 0]; CC1 = [0 1];
 for iter = 1:1:2
        CG0(iter, :) = K0;
        CG1(iter, :) = K1;
        %V-steps
        A0 = AA + BB*K0*CC0;
        A1 = AA + BB*K1*CC1;
        Q0 = sdpvar(2,2); Q1 = sdpvar(2,2);
        H0 = [Q0 A0'*Q0 A1'*Q1; Q0*A0 inv(P0(1,1))*Q0 zeros(2); Q1*A1 zeros(2)
inv(P0(2,1))*Q1];
        H1 = [Q1 A0'*Q0 A1'*Q1; Q0*A0 inv(P0(1,2))*Q0 zeros(2); Q1*A1 zeros(2)
inv(P0(2,2))*Q1];
        lmisysV = [H0 > 0, H1 > 0, Q0 > 0, Q1 > 0];
        solvesdp(lmisysV);
        Q0 = double(Q0);
        Q1 = double(Q1);
        %K-steps
```

```
        a = sdpvar(1, 1);
        K0 = sdpvar(1, 1);
        K1 = sdpvar(1, 1);
        A0 = AA + BB*K0*CC0;
        A1 = AA + BB*K1*CC1;
        M0 = [Q0 A0'*Q0 A1'*Q1; Q0*A0 inv(P0(1,1))*Q0 zeros(2); Q1*A1 zeros(2)
inv(P0(2,1))*Q1];
        M1 = [Q1 A0'*Q0 A1'*Q1; Q0*A0 inv(P0(1,2))*Q0 zeros(2); Q1*A1 zeros(2)
inv(P0(2,2))*Q1];
        lmisysK = [M0 > 0, M1 > 0];
        c = [1 0 0];
        x = [a; K0; K1];
        h = c*x;
        solvesdp(lmisysK,h);
        K0 = double(K0);
        K1 = double(K1);
        %delta-steps
        P0 = P0 + delta;
        if (isequal(P0, P) == 1)
                break;
        end
end
```

B.2. SIMULINK® BLOCK DIAGRAMS FOR THE BANDWIDTH ALLOCATION

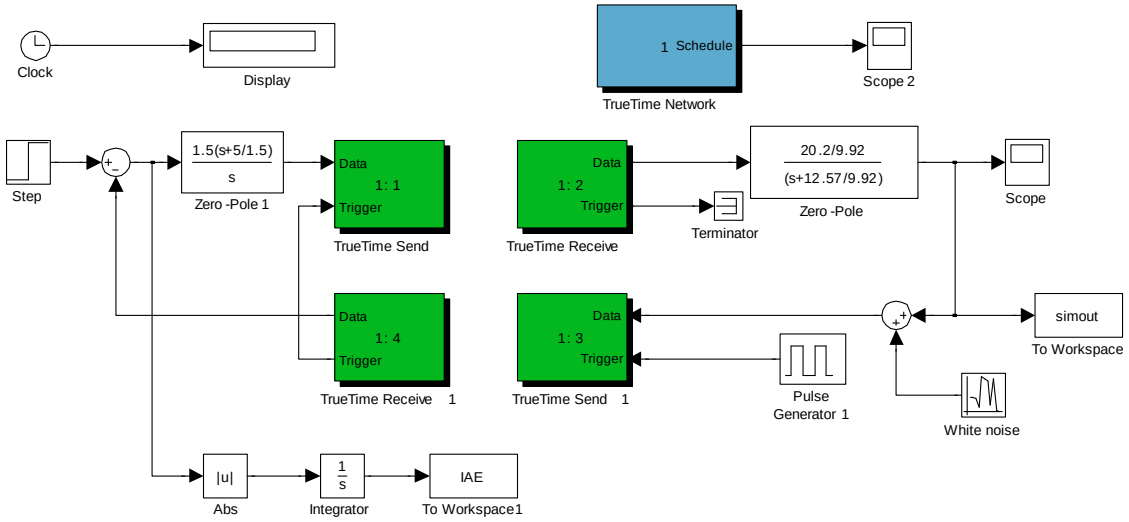


Fig. B.1. A Simulink® block diagram for the bandwidth allocation simulation with single DC motor

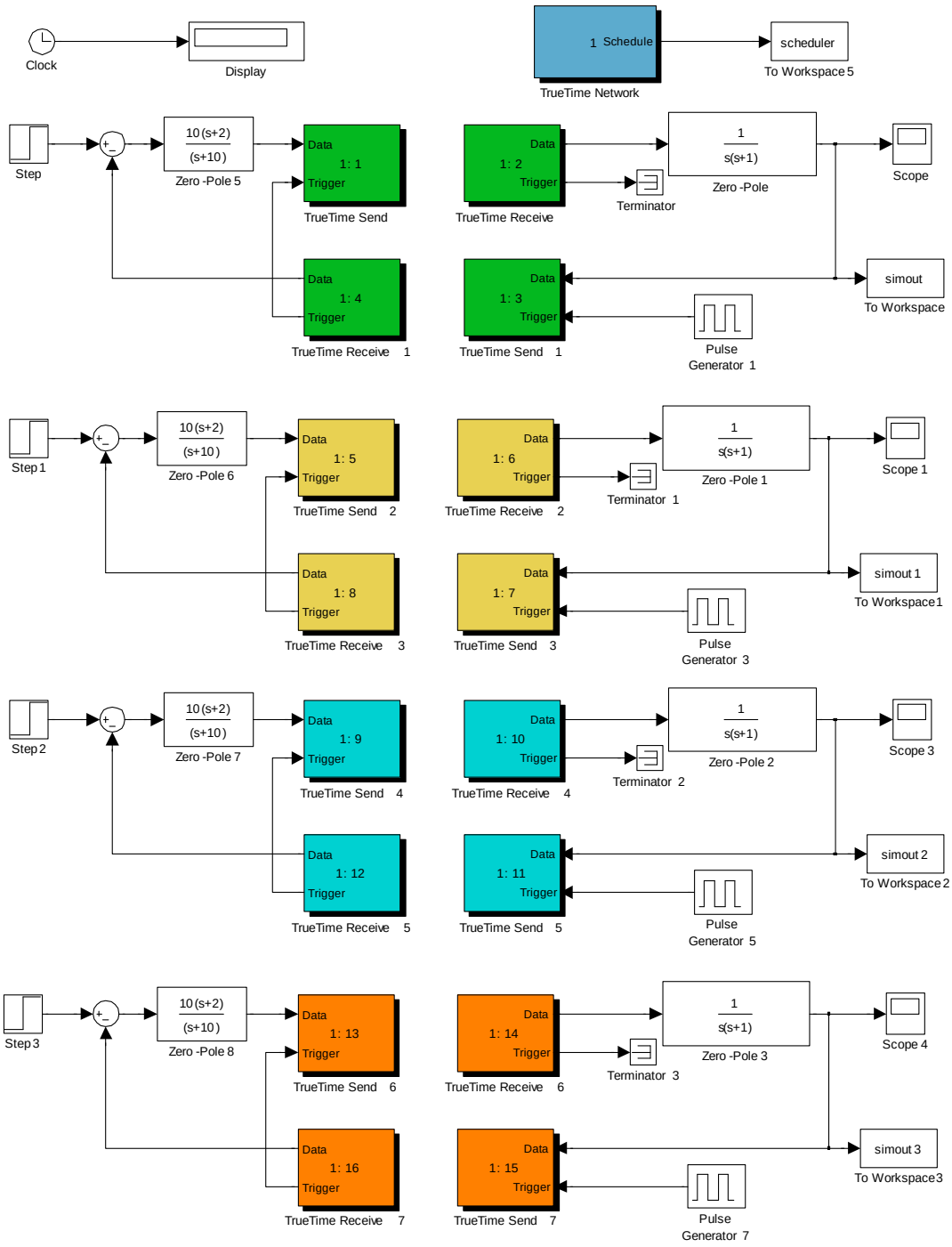


Fig. B.2. A Simulink® block diagram for the bandwidth allocation simulation with four DC motors