Real-Time Operating Environment for Networked Control Systems

Won-jong Kim, *Senior Member, IEEE*, Kun Ji, *Student Member, IEEE*, and Ajit Ambike

***Abstract*—This paper presents the development of a novel real-time operating environment for networked control systems (NCSs). An open-loop unstable magnetic-levitation (maglev) test bed was constructed and used to develop an NCS with real-time application interface (RTAI). A client–server architecture on a local-area network was developed with the network communication based on the user datagram protocol. The control loop of our NCS is closed over the network. This NCS structure gives the best flexibility and has significant economical merits. The implementation of an event-driven server and a time-driven client presented in this paper facilitates a simple timing scheme that does not require clock synchronization between the client and the server. A novel prediction scheme with a multiple-step-ahead control-signal generator is used to maintain system stability in the presence of excessive time delays and data-packet losses in the communication network. The performance of this NCS, based on the predictor algorithm, is demonstrated experimentally. The current system can compensate for up to 20% data-packet losses without losing stability with the maglev real-time-control test bed in the communication network. Our real-time operating environment also improved the command-following capability by a factor of 4 in terms of command frequency.**

***Note to Practitioners*—With the advancement in the automation industry, the need to perform complex remote operations has grown. Ever-increasing computational capabilities and advancements in the networking technology have aided researchers to develop architectures to implement control from a distance. In large-scale control applications of the modern industry, the functional agents, such as sensors, actuators, and controllers are geographically distributed. For smooth working of a control application, all of the agents have to exchange information through communication media. The results of this paper can help to design practical real-time networked control systems in modern industry.**

***Index Terms*—Networked control system (NCS), packet loss, real-time operating environment, time delay.**

I. Introduction

With the advancement in the automation industry, the need to perform complex remote operations has grown. Ever-increasing computational capabilities and bandwidths in the networking technology enabled researchers to develop networked control systems (NCSs) to implement distributed control schemes from a distance. Unlike traditional control systems, an NCS essentially comprises multiple nodes communicating with each other over communication networks. A real-time operating environment is needed in the implementation of an NCS to handle the timings of various events in the communications among these nodes. The success of an NCS relies on the effective integration of computing resources, communication network, and control algorithms in the various levels of its implementation. Several factors, such as time resolution and capability of multi-threading and periodic tasks, affect the selection of an appropriate real-time computing environment.

A substantial amount of work has been done in the area of NCSs. NCSs can be roughly classified into three modes: 1) teleoperation; 2) supervisory control; and 3) feedback control over network. Teleoperation was the first form of an NCS that became popular. Internet-based teleoperation was used in telerobotics, remote manufacturing, telesurgery, and distant education. Hu *et al.* discussed the use of cooperative Internet robots with an interactive human–machine interface [1]. Mitsuishi *et al.* developed a master–slave-type telemicrosurgical system with an intelligent user interface [2].

With supervisory control, a user can give symbolic or analogic instructions remotely to a computer attached to the manipulator instead of remotely guiding the telemanipulator. Luo *et al.* used a supervisory control technique to develop a desktop rapid-prototyping system [3]. Garcia *et al.* developed a telerobotic system using supervisory control based on a hybrid control approach [4]. Srivastava and Kim discussed the supervisory control via the Internet of a ball magnetic-levitation (maglev) system [5]. The setup was based on client/server architecture with an interface programmed with hypertext markup language (HTML) and common gateway interface (CGI).

In teleoperation and supervisory control, the access to the control system is provided to the user in the form of an interface. The controller is implemented on the controlled-environment side of the network (i.e., the control loop is closed locally).

On the other hand, in the feedback control over network, the user side of the network runs the control algorithm. All components, including sensors, actuators, and controllers, are assumed to be interconnected via a communication network. The control loop of the system is closed over the network using protocols to multiplex data from the sensor to the controller and from the controller to the actuator. A client-server-based architecture of feedback control over distributed networks is depicted in Fig. 1. The feedback control loop shares the communication medium with other data-transfer processes. The main advantage of this configuration is flexibility. It allows placing the controller on the network, separated from the controlled processes, without

Manuscript received December 31, 2004; revised March 29, 2005. This work was recommended for publication by Associate Editor A. Monti and Editor M. Wang upon evaluation of the reviewers' comments. This work was supported by the National Science Foundation under Grant CMS-0116642.

W.-J. Kim and K. Ji are with the Department of Mechanical Engineering, Texas A&M University, College Station, TX 77843-3123 USA (e-mail: wjkim@tamu.edu).

A. Ambike is with NuView, Inc., Houston, TX 77079 USA.

Digital Object Identifier 10.1109/TASE.2005.862146

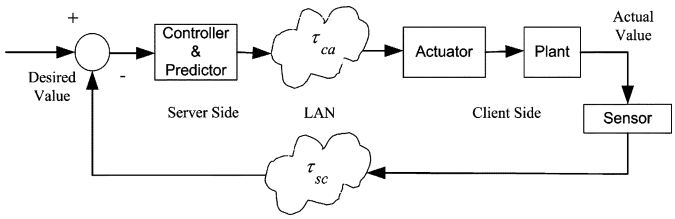

Fig. 1. Block diagram of feedback control over network. The independent network delays from the controller to the actuator and from the sensor to the controller are denoted as τ_{ca} and τ_{sc}, respectively.

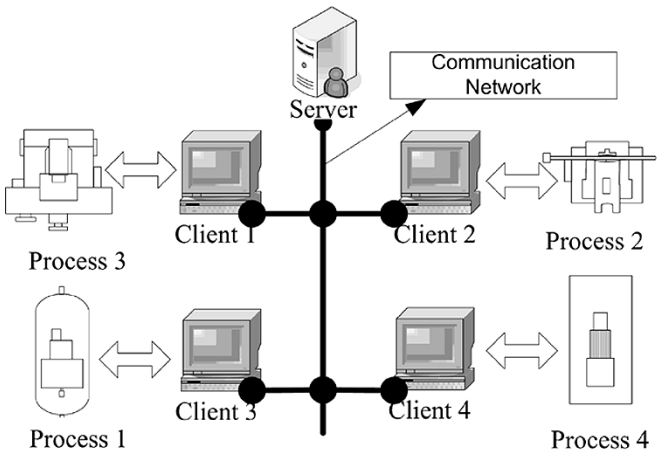

Fig. 2. Distributed control system with multiple clients.

requiring closing the control loops locally with individual computing units. Thus, it has significant economic merit. One application of this NCS mode is a distributed control system as shown in Fig. 2.

The automation industry will soon reap the benefits of high-performance closed-loop control on distributed networks along with the development of reliable high-bandwidth networks and new protocols, such as Internet II and Rether (real-time Ethernet) [6], [7]. Although this field is relatively new and still in its infancy, it has captured significant interest from many researchers worldwide. Bluetooth technology was used to develop wireless control of a rotating inverted pendulum by Eker and Cervin [8]. Ploplys *et al.* concluded that the UDP, an unreliable but faster protocol, was better suited for real-time control over a dedicated wireless computer network [9].

In the development of NCSs, dealing with network delays and data-packet losses in the communication networks has always been a key issue. Ferrell was among the first to work on the problem of time delays in teleoperation [10]. Later, Anderson and Spong studied the bilateral control of the teleoperated system with time delays [11]. That research focused on maintaining the stability in force-reflecting bilateral teleoperators in the presence of substantial time delays. Using the passivity and scattering theory, a criterion was developed to show the usefulness of the bilateral control law. Conway *et al.* introduced new concepts called time and position clutches to deal with the time delays due to telemetry and signal propagation in teleoperations [12].

The development of a real-time operating environment enabling closed-loop real-time control over distributed network is the main focus of this paper. Various key issues regarding the realization of such systems are identified and addressed. An NCS with closed-loop control of a maglev test bed is implemented on an LAN in our real-time operating environment. This NCS also incorporates a novel predictor-based algorithm to stabilize the system in the event of consecutive network delays and data-packet losses. The software architecture is based on the UDP. In the past, dedicated networks were used to study closed-loop real-time control over the distributed network [8], [9]. To accommodate the effect of the cross traffic of packets on the communication-delay profiles, an Ethernet was used for the NCS in the current research. Thus, in our NCS framework presented in this paper, the network is not necessarily dedicated to the NCS. A novel multistep-ahead-predictor algorithm has been developed to stabilize the NCS in the event of network delays and data-packet losses.

The need for real-time operating environments is elaborated in Section II. A brief discussion on various factors affecting the selection of real-time operating environments is given in Section III followed by a review of RTLinux, RTAI, and Linux Control and Measurement Device Interface (Comedi). The development of the software architecture and the implementation of the predictor-based algorithm are presented in Section IV. In this section, a brief overview of the maglev test bed and the extension of our real-time environment for multiclient cases are also given. The experimental verification of the functionality of the NCS and its real-time operational environment is provided in Section V.

II. Need For Real-Time Operating Environments

The Ethernet is widely used in the information-technology (IT) industry. Various communication protocols, such as transmission control protocol/internet protocol (TCP/IP) are used. However, the use of an LAN in NCSs poses several technical challenges including dealing with network latencies. To understand the latency issues, let us consider a client-server communication on an Ethernet as shown in Fig. 3. The client requests some information from the server, and the server responds to that request. The maximum delay in this communication (i.e., the latency) is a significant real-time constraint on the communication system. Table I gives the nomenclature of the time-delay components shown in Fig. 3. We assume that there is no data collision on the network.

As shown in Fig. 3, a typical client–server communication process in an NCS is periodic with a sampling period T. The entire communication process is required to be completed in one period. The total communication time or the latency T_{total} is given by

$$T_{\text{total}} = TC_{\text{prep}} + TC_{\text{wait}} + TC_{\text{transmit}} + TS_{\text{process}} + TS_{\text{prep}} + TS_{\text{wait}} + TS_{\text{transmit}} + TC_{\text{process}}. \quad (1)$$

For an NCS with the architecture shown in Fig. 1, the client side hosts the test bed. The server side implements the controller. The request message sent by the client to the server can carry the sensor data, and the response message sent by the server to the client can carry the control data. The creation of the message to be sent to the server is a time-driven process. Taking

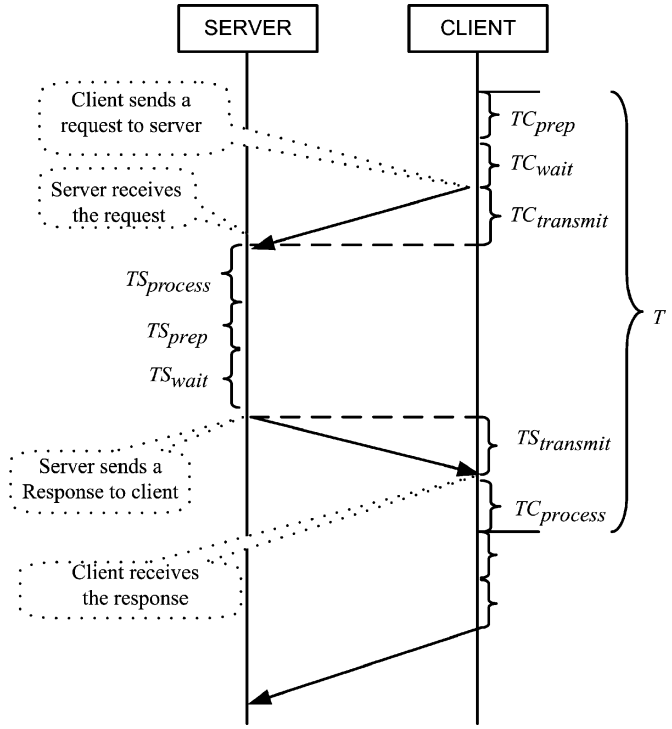

Fig. 3. Time-delay components of the network latency in a periodic client–server communication process.

TABLE I
NOMENCLATURE OF TIME-DELAY COMPONENTS

Symbol	Description
TC_{prep}	Time taken by the client to prepare the request message
TC_{wait}	Time spent by the client waiting for network access
$TC_{transmit}$	Transmission time from the client to the server
$TS_{process}$	Time taken by the server to process the request
TS_{prep}	Time taken by the server to prepare the reply message
TS_{wait}	Time spent by the server waiting for network access
$TS_{transmit}$	Transmission time from the server to the client
$TC_{process}$	Time taken by the client to process the reply
T	Total period of the process on the client side

our maglev setup for an example, it is the responsibility of the operating system (OS) to ensure that a sensor-data sample is taken every 3 ms for the 333.333-Hz sampling frequency. On the other side of the communication network, the server waits for the request message from the client. As soon as it receives a message from the client, it processes the request and creates a reply, the control data in the current research. The moment of the creation of this reply to be sent to the client is dependent on the arrival of the request. In other words, it is an event-driven process. After the server sends back the reply message to the client, the client receives it and generates the control signal. In the beginning of the next sampling period, another sensor-data sample is taken, and a similar communication process takes place and is repeated.

In practice, all of these events have certain deadlines. If some of these deadlines were missed, the stability of the NCS might be negatively affected. In the current research, the maglev test bed to be described in Section IV is open-loop unstable. For our test bed, Srivastava and Kim [5] demonstrated that all of the calculations in the feedback control loop should be completed within 1.4 ms. If this deadline is missed due to the network latency, the system becomes unstable. Moreover, in order to ensure that the time-sensitive events occur at precise times, a real-time operating environment is needed. A real-time system can be defined as a system that responds to externally generated stimuli within a finite and specified period of time [13].

III. SELECTION OF REAL-TIME OPERATING ENVIRONMENTS

A. Factors Affecting Selection of Operating Environments

After the nature of closed-loop NCSs was studied, it was concluded in the previous section that an appropriate OS should be required to successfully implement the distributed architecture. The following factors were considered in the selection of the OS in the context of our maglev test bed.

1) Periodic Tasks: The OS should allow execution of periodic tasks, such as sampling data, sending control data, and receiving feedback data.

2) Time Resolution: It was desirable that the time resolution be as small as 14 μs, which is 1% of the minimum time to close the control loop.

3) Threads: Implementing the algorithms for closed-loop real-time control over communication network involved simultaneous sampling and actuation. Thus, there is a need for multithreaded programming.

B. Prevailing Operating Systems are Inadequate

Commercially available OSs, such as Windows and various versions of Unix and Linux, could be immediate choices. Unfortunately, they are not real-time OSs and their performances with reference to the above-mentioned factors turned out to be dissatisfactory. Two timing tests, as stated by Volz [13], were performed to observe their performances and are described in the following two paragraphs.

The smallest amount of time that can be precisely measured on an OS is known as its clock resolution. It depends upon how the clock is implemented as hardware or software and the method used to access it. There are several low-resolution and high-resolution clock-access commands in the above mentioned OSs. For example, the time required to access the clock in Redhat Linux 7.3 is approximately 0.01 μs whereas its clock resolution is 10 μs. Because the time for the clock read is less than the resolution, many consecutive clock reads can be made before the value returned by the clock changes. This principle was used in the first timing test. The number of times the clock was accessed before the change in value returned by the clock-access function was recorded in an array over several iterations. Fig. 4 represents the results of the first timing test on Windows 2000 and Redhat Linux 7.3. These two OSs are implemented in two separate PCs with the same 1.7-GHz Pentium IV processors. The programs created in these two OSs are both on user level and set with the highest priorities. Significant variations in these plots indicate the existence of nondeterministic OS activities. This simple test demonstrated the non-real-time characteristics of the two popular OSs.

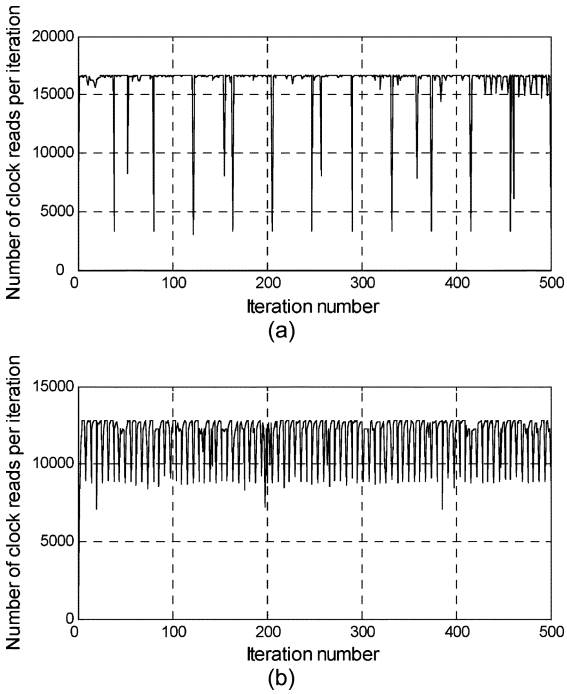

Fig. 4. Plots of the number of clock reads per iteration for the first timing test on (a) Windows 2000 and (b) Redhat Linux 7.3.

When the value returned by the clock-access function changes, the difference between this new returned value and the previously returned value is the clock resolution. In the second timing test, the clock resolution was calculated and plotted over many iterations. There were variations in the values returned as clock resolution. Thus, the maximum value is considered to be the clock resolution for the corresponding OS. Fig. 5 presents the results of the second timing test on Windows 2000 and Redhat Linux 7.3, respectively. Whereas the clock resolution of Redhat Linux 7.3 was found to be uniform, that of Windows 2000 was not.

C. Real-Time Operating Solutions

The Linux kernel provides several sophisticated services such as hardware management, event polling, peripheral interrupts, scheduler classes dealing with priorities, interprocess communication, and implementation of network protocols such as TCP/IP. However, Linux alone as an OS lacks real-time support. It is necessary to make some modifications in its kernel behaviors such as interrupt handling and scheduling policies to make it a real-time platform with low latency and high predictability of timing performances. Based on Linux OS, there are two real-time operating solutions.

1) Real-Time Linux (RTLinux): RTLinux was developed at the New Mexico Institute of Technology by Barabanov and Yodailen [14]. The RTLinux scheduler treats the Linux OS kernel as an idle task. Linux only executes when there are no real-time tasks to run and the real-time kernel is inactive. The software emulation of interrupt control hardware makes it possible that the Linux task can never block interrupts or prevent itself from being pre-empted. RTLinux has been designed so that the real-time kernel never waits for the Linux to release any resources. The real time-kernel does not directly request memory, share

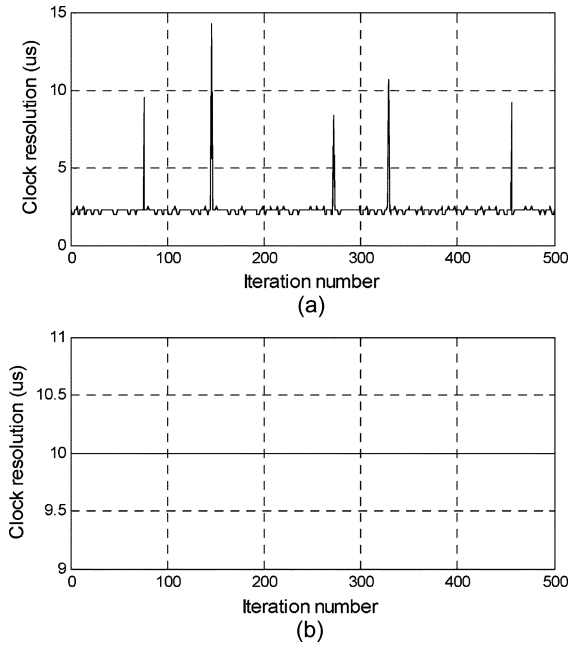

Fig. 5. Plots of the clock resolutions obtained for the second timing test on (a) Windows 2000 and (b) Redhat Linux 7.3.

TABLE II
RTAIs TYPICAL PERFORMANCE

Context switch time	4 μs
Interrupt response	20 μs
Maximum periodic task rate	100 kHz
One-shot task rate	30 kHz

spin locks, or synchronize data structures, except in tightly controlled situations.

2) RTAI: RTAI was developed as a real-time operating environment solution at Dipartimento di Ingeneria Aerospaziale Politecnico di Milano (DIAPM) [14]. Based on the Linux kernel, it provides deterministic and pre-emptive performance in addition to allowing the use of all standard Linux drivers, applications, and functions. RTAI modifies the Linux kernel to make it a real-time operating environment. RTAI offers the same services as the Linux kernel core, adding the features of a real-time OS (RTOSs). Compared to the commercially available real-time OSs, The RTAIs performance is very competitive with the best commercial RTOSs such as VxWorks, QNX, etc. [14]. Table II summarizes the typical performance of RTAI. RTAI is open source and free under the terms of the GNU (GNU is not Unix) lesser general public license.

Fig. 6 shows the results of the same two timing tests as in Section III-B run on RTAI 24.1.12 with Redhat Linux 7.3. In Fig. 6(a), the straight line indicates that there were no significant indeterministic OS activities. Although the spikes in Fig. 6(b) denote the variation in the clock resolution from 1 to 4.1 μs, the clock resolution reported is consistently less than 5 μs. This allowed us to measure the time intervals as small as 5 μs, which is much less than the 1% criterion (14 μs) set in Section III-A-2. In the next section, we develop a novel real-time operating environment based on RTAI with Linux that provides a competitive solution to NCSs.

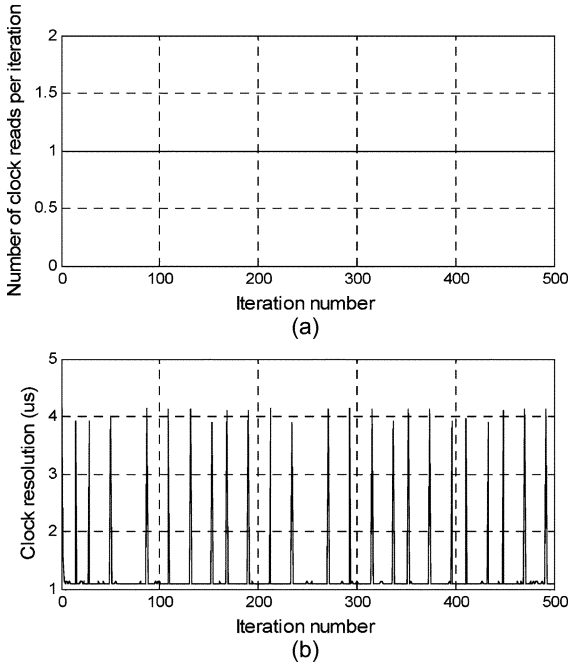

Fig. 6. Plots showing the results of the timing tests on RTAI 24.1.12 with Redhat Linux 7.3. (a) Number of clock reads per iteration for the first timing test. (b) Clock resolutions obtained for the second timing test.

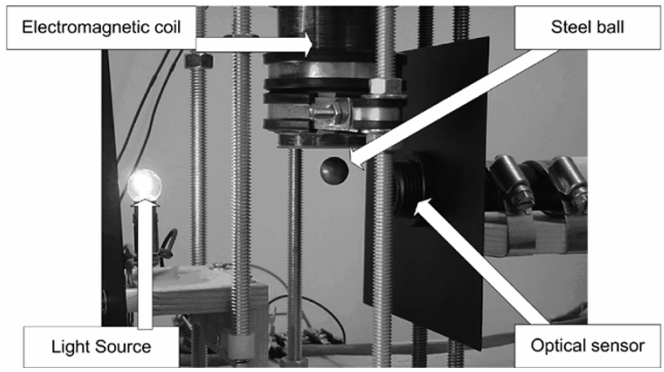

Fig. 7. Single-actuator ball maglev system [15].

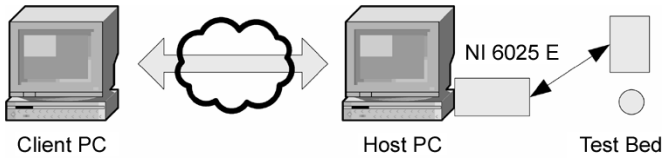

Fig. 8. Block diagram of the developed distributed NCS.

IV. Development of the NCS Architecture

A. Test Bed

The single-actuator ball maglev system, developed by Paschall [15] was used as the test bed for this research. As shown in Fig. 7, a steel ball with the diameter of 12.7 mm is held in stable levitation at a steady-state operating position. An electromagnet is used to produce the force to support the ball against gravity. The force is produced by controlling the current in the coil of the electromagnet using a control system implemented on a PC. The digital controller designed by Srivastava snd Kim [5] to stabilize this maglev system is given by

$$D(z) = 4.15 \times 10^4 \left[\frac{z^2 - 1.754z + 0.769}{z^2 - 0.782z - 0.13} \right]. \qquad (2)$$

National Instruments' PCI-6025E is the data-acquisition (DAQ) board for the experiments. Comedi [16] was used to develop the hardware–software interface. Comedi is a free software project for tools, libraries, and drivers for various forms of DAQ and provides a collection of drivers for a variety of common DAQ plug-in boards that use either a peripheral component interconnect (PCI) or peripheral component microchannel interconnect architecture (PCMCIA) bus. It works with the standard Linux kernel as well as the real-time extensions such as RTLinux and RTAI. The single-core module called Comedi provides common functionality, and the DAQ-board-specific driver module provides low-level functionality. Comedilib provides a developer-friendly interface to the Comedi devices.

B. Components and Protocols

The distributed NCS as shown in Fig. 8 was developed. A desktop PC with a 600-MHz Celeron processor with the implemented controller acts as the Server PC. Another desktop PC with a 1.7-GHz Pentium IV processor with the PCI-6025E board was used for DAQ and acts as the Client PC. Linux with RTAI is implemented on both PCs. A 100-Mb/s Ethernet LAN was used as the medium of communication. This arrangement has the advantage of using commercial off-the-shelf hardware. It might not be strictly necessary to use a real-time OS on the server side as long as it can ensure the time-constrained control signal calculation and transmission. Since in our design the server controller is event driven, it also has time-constrained events. Thus, using real-time OS on both the sever side and the client side can cause less time delay and make the communication between the sever and the client more efficient and compatible.

The network communication was originally based on the TCP/IP protocol suite, and the programs were developed in the C programming language [17]. Functions provided by the sockets application programming interface (API) were used for implementation. The TCP/IP suite provides various ways of data transfer. Although the TCP is a very reliable protocol for communication over computer networks, it consumes more computing resources, such as CPU time and memory, and introduces significant time delays in the communication [18]. For closed-loop control over the network, the added reliability provided by the TCP may not worth the cost of the network delays it introduces [9]. On the other hand, the UDP does not provide additional services, such as ensuring ordered data delivery and robustness, as provided by the TCP and is less reliable. Yet, the UDP has fewer overheads and induces fewer network delays compared to the TCP. Reduced round-trip time delays justified the use of the UDP for NCS applications. In this paper, we intentionally introduce packet losses to demonstrate the effectiveness of our real-time control algorithm. We even introduce packet losses with up to a 20% drop rate. For this purpose, the TCP is a costly protocol and not suitable for our research. Thus, the UDP is used in this research.

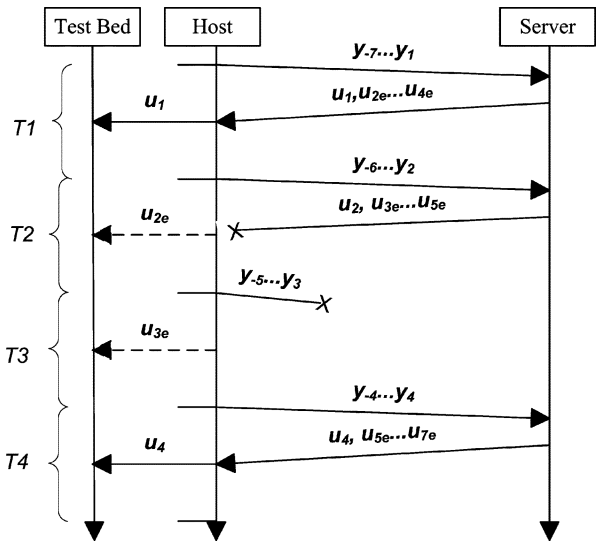

Fig. 9. Example timing diagram of the NCS communication. An arrow with a cross tip denotes that the data packet was lost. In the current architecture, lost data are not more detrimental to the system stability or performance than late data.

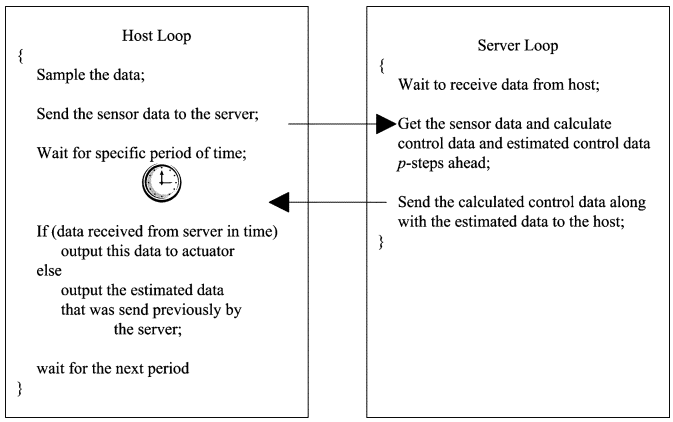

Fig. 10. Psuedocode for the client–server communication.

C. Timing of Events

In the past, synchronization of clocks was tried to coordinate the events in the networked control loop [19]. It is a complicated process, generating additional network traffic to deliver the synchronized clock signals, and also requires continuous adjustment to retain this synchronization once generated. A comparatively simpler approach is presented in this paper.

Our networked control architecture does not involve clock synchronization and is based on a combination of time-driven and event-driven communicating processes. Fig. 9 shows an example timing diagram of network communication between the client and the server. The communications labeled y denote the sensor data transferred from the client to the server, and the communications labeled u, the control signal data transferred from the server to the client. The subscripts of these labels denote the sampling-period indices $(-7,\ldots,7)$ and indicate whether the data are an estimate (e). For example, y_2 is the sensor data of the second sampling period, u_3 is the control signal data for the third sampling period, and u_{2e} is the control signal estimate for the second sampling period.

Fig. 10 shows a pseudocode for the execution of the closed-loop control over the LAN. Sampling and actuation take place on the client side and are time-driven whereas calculation of the control signal is event-driven. It was calculated that if the sampling period is 3 ms, the actuation has to occur within 1.4 ms after sampling for the system stability [12]. Thus, for the 333.333-Hz sampling frequency, sampling and actuation are two cyclic executions offset by 1.4 ms on the client side. Thus, in the software, sampling and actuation are implemented in two different periodic threads. The sampling thread samples the data and sends them using a UDP socket. After sampling has occurred, the actuation thread waits for 1.4 ms for the arrival of the data on the same UDP socket. If the control data arrive in time, they are used immediately for actuation. If not, the estimate of the control data that the server sent in previous messages will be used. A discussion on this estimation of control data is given in the following.

The control signal is predicted on the server side and is event-driven. The process on the server side waits for the arrival of the sensor data from the client. As soon as the sensor data arrive, an appropriate control signal is calculated and sent using a UDP socket. In addition to the current control data, the predicted control data for the next p sampling periods are sent to the client. The criterion to select p will be discussed in the following sections. To calculate these estimates, parametric predictors are used. These predictors require past sensor data that are transmitted to the server along with the current sensor data.

The client stores and updates the values of these predicted control data after each data-packet arrival from the server. In the event of time delays and data losses, these estimated values are used to generate control signals to stabilize the system. Another option might be to output zero value, when the actuator receives no control action [9]. With an open-loop unstable plant such as our current maglev test bed, however, the stability of the system will be eventually lost in the event of excessive time delays. The multiple-step-ahead prediction is used to maintain system stability.

D. Predictor-Based Control Strategy

As shown in Fig. 9, the server calculates the predicted controls for the next p sampling periods in addition to the control for the current sampling period. Accurate prediction of control data is important to guarantee the stability of the system. It is also necessary to choose an appropriate order for the predictor. The accuracy of prediction and the number of computations required for prediction are the two important factors to consider in the decision of the order of the predictor. A tradeoff exists between the accuracy of a predictor and the number of computations done for each prediction. In general, higher order predictors have a better accuracy of prediction than lower-order ones of the same type. On the other hand, higher-order predictors take more computational time for a prediction than lower-order ones. A good predictor should give reasonably accurate prediction with the minimum usage of computational resources.

Another factor to be considered while deciding the order of predictors is the amount of past data required. With higher order

TABLE III
BEST FITS FOR AR MODELS

Order	1-step	2-step	3-step	4-step	5-step	6-step	7-step	8-step
8	74.43	67.02	65.40	65.60	64.78	63.98	64.86	64.39
7	72.63	64.79	63.76	63.98	62.99	60.96	60.36	60.04
6	72.05	62.68	61.35	62.16	61.27	59.67	57.34	55.92
5	71.95	62.86	61.74	62.05	60.43	59.61	57.64	56.97

predictors, more past data are required for computation. In any communication architecture, however, there is practical limitation. In the current architecture, the storage of past sensor data and the control data is done on the client side. If the server had to store these data, the data on the server side would be incomplete in the event of packet drops from the client to the server. Therefore, the client passes the past sensor data to the server each time along with the current sensor data. It is necessary that these data travel in a single data packet. If it is fragmented into multiple packets and any of these fragmented packets is lost, the past data required for prediction by the server would be incomplete.

IP provides packet delivery service for protocols such as UDP and TCP. The version of IP used in the current research is Internet Protocol, version 4 (IPv4). A minimum reassembly buffer size identified by IPv4 is 576 B [20]. This is the minimum datagram size that is guaranteed to be supported by any implementation of IPv4. In the event of congestion, UDP packets are lost. Larger packets have a higher probability of getting lost than the smaller ones. Thus, it is required to keep the size of packets to a minimum while sending an adequate amount of data for better prediction.

To collect online sensor data of the stable system, the test bed was operated with the feedback loop closed locally. The sensor data of the system for 17 162 samples were collected. MATLAB used half of these data to develop predictors and the remaining half for their validation. The percentage of the sensor variations reproduced by the predictors is known as the best fit [21]. Higher best fit implies better prediction. Table III represents the best-fit values for autoregressive (AR) models for various step-ahead predictors.

Based on the above results, an eighth-order predictor was designed after numerous design iterations. Predictors were also designed for up to four-step-ahead predictions of the sensor data. The four-step-ahead prediction for the control signal was calculated using the predicted sensor data. The parameter vectors for various predictors, calculated using MATLAB, were used to develop the following prediction equations for our maglev test bed:

$$\begin{aligned}\hat{y}(t+1) =& 0.8122y(t)-0.3479y(t-1)-0.0294y(t-2)\\ &+0.4605y(t-3)+0.0742y(t-4)\\ &+0.1042y(t-5)+0.1117y(t-6)\\ &-0.3561y(t-7) \qquad (3)\\ \hat{y}(t+2) =& 0.3117y(t)-0.3119y(t-1)+0.4366y(t-2)\\ &+0.444\,82y(t-3)+0.1645y(t-4)\\ &+0.1964y(t-5)-0.2653y(t-6)\\ &-0.2892y(t-7) \qquad (4)\end{aligned}$$

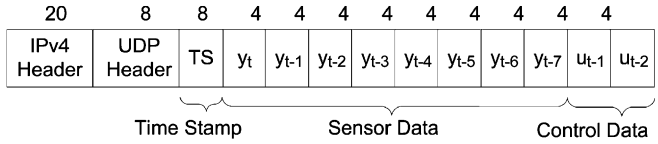

Fig. 11. Composition of an IPv4 packet transmitted from the client to the server.

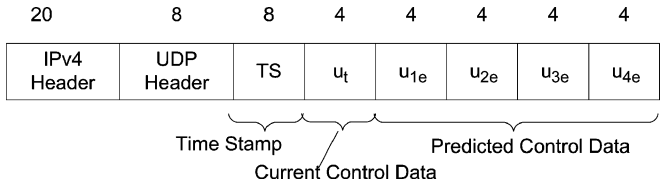

Fig. 12. Composition of an IPv4 packet transmitted from the server to the client.

$$\begin{aligned}\hat{y}(t+3) =& -0.0587y(t)+0.3281y(t-1)+0.4390y(t-2)\\ &+0.3080y(t-3)+0.2195y(t-4)\\ &-0.2329y(t-5)-0.2544y(t-6)\\ &-0.1110y(t-7) \qquad (5)\\ \hat{y}(t+4) =& 0.2804y(t)+0.4594y(t-1)+0.3097y(t-2)\\ &+0.1925y(t-3)-0.2372y(t-4)\\ &-0.2605y(t-5)-0.1176y(t-6)\\ &+0.0209y(t-7). \qquad (6)\end{aligned}$$

E. UDP Packet Composition

The composition of a typical 76-B-long IP packet transmitted from the client to the server is shown in Fig. 11. It consists of a 20-B-long IP header, an 8-B-long UDP header, an 8-B-long time stamp, eight 4-B-long sensor-data values, and two 4-B-long previous control-data values. A time stamp is taken on the client side at sampling and is sent to the server. The server does not modify the time stamp but sends it back to the client along with the calculated control data. This time stamp is then used by the client to identify whether the arrived data packet is the expected packet or a delayed packet. If a packet is delayed, it is simply discarded in the current scheme.

The composition of a typical 56-B-long IP packet transmitted from the server to the client is shown in Fig. 12. It consists of a 20-B-long IP header, an 8-B-long UDP header, an 8-B-long time stamp, one current control-data value, and four predicted control-data values.

F. Extension for Multiclient Cases

Our real-time operating environment configuration and predictor-based control strategy for NCSs can be extended for multiclient cases as shown in Fig. 2. Ethernet uses the carrier-sense multiple access with collision detection (CSMA/CD) mechanism to resolve contention on the communication medium. The CSMA/CD protocol is specified in the IEEE 802.3 network standard and is described briefly as follows [18].

In the proposed NCS configuration, the server controller is event-driven, and it works only after it receives the sensor data from the client. By simply implementing a first-in-first-out

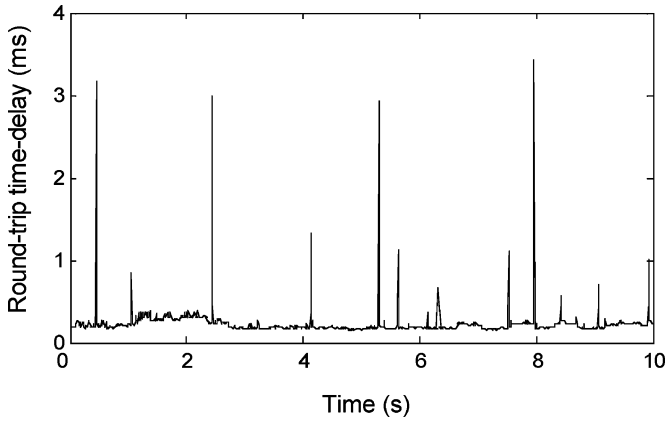

Fig. 13. Round-trip time delays of the maglev setup.

(FIFO) queue on the sever side, there is no control-data collision in the transmission path from the server controller to the clients. However, in the transmission of sensor data from the client sensors to the server controller, data collision maybe a problem, and the network load depends on the sampling frequencies of these client sensors.

1) Low Network Load Case: In Fig. 2, if the sampling frequencies of these clients are very low and distinct, the clients send their sensor data sporadically with different periods. Thus, the possibility of data collision is very small. Our predictor-based control strategy can be used to ensure the system stability when data collision occurs occasionally. Experimental results in the following section verify that our control strategy can ensure system stability even with up to 20% data-packet losses.

2) Heavy Network Load Case: In Fig. 2, if the sampling frequencies of these clients are very high and close, more than one client might send their sensor data simultaneously. Several solutions have been proposed for data collision problems for this case. Some try to prioritize CSMA/CD (e.g., LonWorks) to improve the response time of critical packets [22]. Using switched Ethernet by subdividing the network architecture is another way to increase its efficiency [18]. These aspects are out of the scope of this paper.

V. Experimental Verification of Proposed NCS and Its Real-Time Operating Environment

To verify the effectiveness of the proposed NCS architecture and the developed real-time operating environment, three sets of experiments were conducted with the experimental setup shown in Fig. 8. For this setup, Fig. 13 shows a round-trip time-delay profile between the client and the sever. The average round-trip time delay was about 230 μs with its standard deviation of about 200 μs. The sampling period of our system was 3 ms. Thus, we introduced artificial time delays/packet losses if necessary for the purpose of demonstrating the effectiveness of our algorithm developed in Section IV.

A. Real-Time Operating Environment Experiments

1) First Set of Experiments: The performance of our real-time operating environment was tested herein. First, the single-actuator ball maglev system was controlled using a PC with a 1.7-GHz Pentium IV processor and a DAQ board PCI-6025E from National Instruments. Windows 2000 was used as the OS to interact with this hardware. Fig. 14 shows the system response of tracking a sinusoidal position command. The maximum command frequency that this control system could follow was 0.7 Hz.

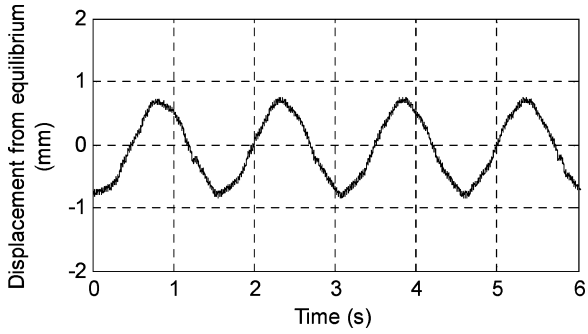

Fig. 14. Response of tracking a sinusoidal command of 0.7 Hz with a non-real-time operating system.

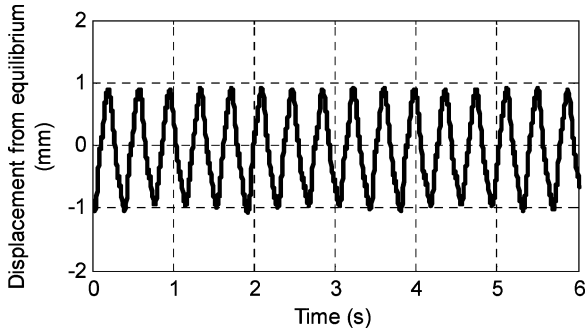

Fig. 15. Response of tracking a sinusoidal command with a frequency of 2.8 Hz in our real-time operating environment.

Then we replaced the control system with Linux RTAI with Comedi. The conditions of the hardware setup were exactly the same as those in the first experiment. The DAQ board was again PCI-6025E. The PC used in this experiment contained the same processor (1.7-GHz Pentium IV) as the one used in the first experiment. Fig. 15 shows the system response of tracking a sinusoidal position command. The maximum command frequency that this NCS could follow was 2.8 Hz. This factor of four improvement in command frequency resulted from the efficiency and the deterministic nature of our real-time operating environment.

B. Performance of NCS Architecture Dealing With Packet Losses Using Proposed Prediction Algorithm

1) Second Set of Experiments: In the second set of experiments, the real-time NCS architecture was tested for the maximum number of allowable consecutive delays. No predicted control was used, a data packet was assumed to be lost at the 3000th sample (at 3000×3 ms $= 9$ s), and no signal was output to the actuator. The system being open-loop unstable, it lost stability immediately as expected, and the levitated ball could not maintain its equilibrium position. The output of the system is shown in Fig. 16(a).

In the second experiment of this set, the control signals based on the prediction of the sensor data were used. Consecutive

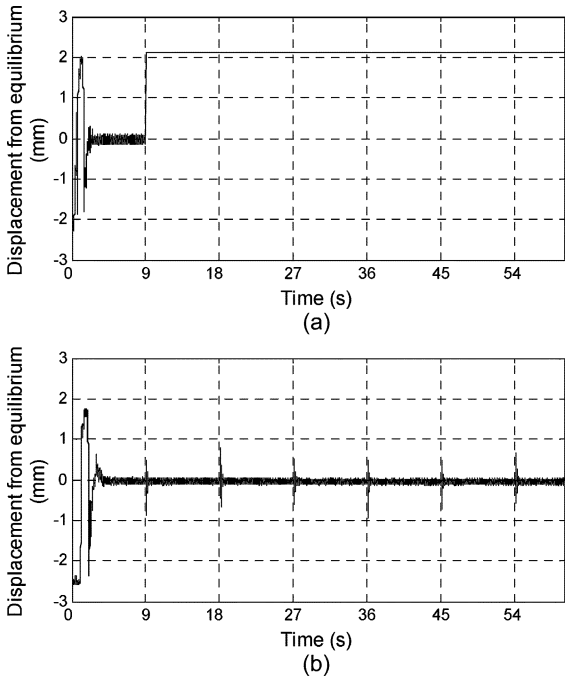

Fig. 16. Plots of the ball displacement from the equilibrium position versus time. (a) System response for the first experiment of the second set. (b) System response for the second experiment of the second set.

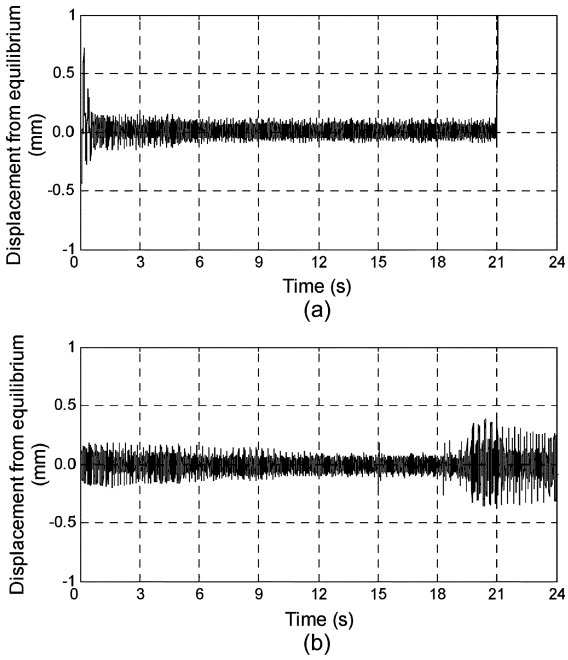

Fig. 17. Plots of the ball displacement from the equilibrium position versus time. (a) System response for the first experiment of the third set. (b) System response for the second experiment of the third set.

packet losses were simulated by rejecting the arrived data p successive times every 3000 samples (i.e., every 9 s). The value of p was increased by one in every run of the experiments starting from zero. The maximum value of p for our maglev test bed was found to be two. In other words, the system accommodated two consecutive packet drops without losing stability. The response of the system is shown in Fig. 16(b). Every time these consecutive delays occurred, the system performance was degraded temporarily.

2) Third Set of Experiments: In the third set of experiments, the real-time NCS architecture was tested for the maximum allowable packet-loss rate. In the first experiment of this set of experiments, no predicted control was used, an artificial delay of 2 ms was introduced with the real-time sleep function on the server side at the 7000th sample (at 7000×3 ms $= 21$ s). Since the data did not arrive in time at the client side, the system lost its stability and the ball could not maintain its equilibrium position. The response of the system is shown in Fig. 17(a).

In the second experiment of this set, an artificial time delay of 2 ms was introduced on the server side for every qth sampling period starting from the 6500th sample (at 6500×3 ms $= 19.5$ s). The value of q was tested for 10, and the system was found to be stable. The value of q was then reduced by one for each subsequent run of the experiment, and the system was checked for stability. The minimum value of q was found out to be 5. This represented the case of one long time delay of sporadic nature in five consecutive delays. Thus, system stability was maintained in the event of up to 20% loss of data packets in communication. The response of the system for q equal to five is shown in Fig. 17(b). In the event of these simulated network delays, the ball did not fall down from its equilibrium position. However, the position fluctuation of the ball about the equilibrium point increased. This dynamic degradation originated from the use of control signals based on the predicted sensor data because actual sensor data were not available due to data-packet losses.

VI. Conclusion

Ever-increasing computational capabilities and network bandwidths enabled researchers to develop NCSs to implement distributed control schemes. A real-time operating environment is essential to properly time key communication events in an NCS and to have complete control on their execution. In this paper, we established a novel real-time environment for NCSs based on RTAI 24.1.12 with Redhat Linux 7.3 after considering several selection factors. The closed-loop control of an open-loop unstable magnetic ball levitation system was demonstrated over a 100-Mb/s Ethernet for its experimental verification. Due to its better real-time performance, the UDP was used in the communication architecture.

Our real-time operating environment improved the command-following capability by a factor of 4 in terms of command frequency. A multiple-step-ahead predictor was implemented wherein predicted control data were used to stabilize the maglev system in the event of time delays and data-packet losses in the communication network. The number of consecutive network delays compensated for by the system depends on the accuracy of the predictors and the system dynamics. The eighth-order predictor designed using an AR model was able to stabilize the maglev test bed for up to 20% data-packet losses in the LAN.

Acknowledgment

The authors would like to thank Dr. R. A. Volz and Dr. S. Jayasuriya for their valuable comments and suggestions.

REFERENCES

[1] H. Hu, L. Yu, P. W. Tsui, and Q. Zhou, "Internet-based robotic systems for teleoperation," *Int. J. Assem. Automat.*, vol. 21, no. 2, pp. 143–151, May 2001.

[2] M. Mitsuishi, S. Tomisaki, and T. Yoshidome, "Tele-micro-surgery system with intelligent user interface," in *Proc. IEEE Int. Conf. Robotics Automation*, vol. 2, San Francisco, CA, Apr. 2000, pp. 1607–1614.

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

[4] C. E. Garcia, R. Carelli, J. F. Postigo, and C. Soria, "Supervisory control of a telerobotic system: a hybrid control approach," *Control Eng. Practice*, vol. 11, no. 7, pp. 805–817, Jul. 2003.

[5] A. Srivastava and W.-J. Kim, "Internet-based supervisory control with stochastic delay models," in *Proc. American Control Conf.*, vol. 1, Denver, CO, Jun. 2003, pp. 627–632.

[6] S. Lankes, M. Reke, and A. Jabs, "A time-triggered Ethernet protocol for real-time CORBA," in *Proc. 5th IEEE Int. Symp. Object-Oriented Real-Time Distributed Computing*, Washington, DC, Apr. 2002, pp. 215–222.

[7] C. Venkatramani, "Design, implementation and evaluation of RETHER: A real-time Ethernet protocol," Ph.D. dissertation, State Univ. New York Stony Brook, Stony Brook, Dec. 1996.

[8] J. Eker and A. Cervin, "Distributed wireless control using bluetooth," in *Proc. IFAC Conf. New Technologies Computer Control*, Hong Kong, China, Nov. 2001.

[9] N. J. Ploplys, P. A. Kawka, and A. G. Alleyne, "Closed-loop control over wireless network," *IEEE Contr. Syst. Mag.*, vol. 24, no. 3, pp. 58–71, Jun. 2004.

[10] W. R. Ferrell, "Delayed force feedback," *IEEE Trans. Human Factors Electron.*, vol. HFE-8, no. 5, pp. 449–455, Oct. 1967.

[11] R. J. Anderson and M. W. Spong, "Bilateral control of teleoperators with time delay," *IEEE Trans. Automat. Contr.*, vol. 34, no. 5, pp. 494–501, May 1989.

[12] L. Conway, R. A. Volz, and M. W. Walker, "Teleautonomous systems: Projecting and coordinating intelligent action at a distance," *IEEE Trans. Rob. Autom.*, vol. 6, no. 2, pp. 146–157, Apr. 1990.

[13] R. A. Volz, Real-Time Computing, Lecture Notes for CPSC 456, Dept. Comput. Sci., Texas A&M University, College Station, TX, 2003.

[14] P. Mantegazza. DIAPM RTAI—Real-time application. [Online] Available: http://www.rtai.org

[15] S. C. Paschall II, "Design, fabrication, and control of a single actuator magnetic levitation system," Senior honors thesis, Dept. Mech. Eng., Texas A&M Univ., College Station, May 2002.

[16] D. Schleef. (2000) Linux Control and measurement device interface. [Online] Available: http://www.comedi.org

[17] A. Ambike, "Closed-loop real-time control on distributed networks," Masters' thesis, Dept. Mech. Eng., Texas A&M Univ., College Station, Aug. 2004.

[18] A. S. Tanenbaum, *Computer Networks*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2001.

[19] L. Zhang, Z. Liu, and C. H. Xia, "Clock synchronization algorithms for network measurements," in *Proc. IEEE INFOCOM*, vol. 1, Jun. 2002, pp. 160–169.

[20] W. R. Stevens, *UNIX Network Programming*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 1998.

[21] L. Ljung and T. Soderstrom, *Theory and Practice of Recursive Identification*. Cambridge, MA: MIT Press, 1983.

[22] M. Neugebauer, J. Plonnigs, K. Kabitzsch, and P. Buchholz, "Automated modeling of LonWorks building automation networks," in *Proc. IEEE Int. Workshop Factory Communication System*, Sep. 2004, pp. 113–118.

Won-jong Kim (S'89–M'97–SM'03) received the B.S. (Hons.) and M.S. degrees in control and instrumentation engineering from Seoul National University, Seoul, Korea, in 1989 and 1991, respectively, and the Ph.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology (MIT), Cambridge, in 1997.

In 2000, he joined the Department of Mechanical Engineering, Texas A&M University (TAMU), College Station, where he is currently Assistant Professor. He was also with SatCon Technology Corporation, Cambridge, MA, for three years. His teaching and research interests focus on the analysis, design, and real-time control of mechatronic systems, networked control systems, and nanoscale engineering and technology. He holds three US patents on precision positioning systems.

Dr. Kim received the Grand Prize from the Korean Institute of Electrical Engineers' Student Paper Contest in 1988. He received the Gold Prize from Samsung Electronics' Humantech Thesis Prize for his 1997 dissertation. He was a semifinalist of the NIST's Advanced Technology Program 2000 Competition. NASA granted him the Space Act Award in 2002 and he was appointed a Select Young Faculty Fellow by TAMU College of Engineering and the Texas Engineering Experiment Station twice in 2003 and 2005. He is the Chair of the ASME Nanoscale Control Technical Panel and a member of the IEEE Nanotechnology Council. He received the Professional Engineering Publishing Award for the best paper published in the 2004 volume of the *Journal of Engineering Manufacture* in 2005. He is a member of ASME, ASPE, KSEA, Pi Tau Sigma, and Sigma Xi.

Kun Ji (S'04) was born in Yangzhou, Jiangsu, China, in 1977. He received the B.S. and M.S. degrees in mechanical engineering from Tsinghua University, Beijing, China, in 1999 and 2002, respectively. He is currently pursuing the Ph.D. degree at Texas A&M University, College Station.

His research interests are networked control system design and real-time control system design.

Ajit Ambike was born in Pusegaon, India, in 1979. He received the B.E. degree in mechanical engineering from Government College of Engineering, Karad, India, in 2000 and the M.S. degree in mechanical engineering from Texas A&M University, College Station, in 2004.

Currently, he is a Software Engineer with NuView, Inc., Houston, TX. His research interests include networked control systems and real-time computing.