Real-Time Operating Environment for Networked Control Systems

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

***Abstract*—This paper discusses the real-time aspects of networked control systems' (NCSs) operating environments. An open-loop unstable magnetic-levitation (maglev) test bed was constructed and used to develop an NCS with a real-time application interface (RTAI) operating environment. A client-server architecture on a local area network (LAN) was developed with the network communication based on the user datagram protocol (UDP). 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 involving the multiple-step-ahead generation of control signals is used to maintain system stability in the presence of excessive time delays and packet losses in the communication network. 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.**

***Keywords*—Networked control system, time delays, packet losses, real-time computing environment.**

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 NCSs to implement control from a distance. The 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 efficient integration of computing resources, communication network, and control algorithms in different levels of the automation industry. A substantial amount of work has been done in the area of NCSs. Teleoperation was the first form of an NCS that became popular. Internet-based teleoperation was used in telerobotics, remote manufacturing, tele-surgery, and distance education [1–4]. Ferrell [5] was the first to work on the problem of time delays in teleoperation. Later, Anderson and Spong studied the bilateral control of the teleoperated system with time delays [6]. Conway *et al.* [7] introduced new concepts called time and position clutches to deal with the time delays. Bluetooth wireless technology was used to develop wireless control of a rotating inverted pendulum by Eker and Cervin [8]. Ploplys *et al.* [9] concluded that the UDP, an unreliable but faster protocol, is better suited for real-time control over dedicated wireless computer networks.

An investigation on real-time operating environments enabling closed-loop real-time control over networks is the main focus of this paper. Various key issues regarding the realization of such systems have been identified and addressed. An NCS with closed-loop control of a maglev setup is implemented on a LAN in a real-time operating environment. The stability of such control systems in the presence of time delays has been an important area of research. A novel multiple-step-ahead predictor-based algorithm was developed to stabilize the NCS in the event of network delays and packet loss.

The need of real-time operating environments is presented in Section II. A brief discussion on the factors affecting the selection of real-time operating environments is given in Section III followed by a review of RTAI and Linux Control and Measurement Device Interface (comedi). The development of software architecture and the implementation of the predictor-based algorithm are presented in Section IV. In Section V, the experimental verification of the NCS and its real-time operational environment is given.

II. NEED OF REAL-TIME OPERATING ENVIRONMENT

The Ethernet is widely used in the Information Technology (IT) industry and allows Internet connection. Various standard communications protocols such as Transmission Control Protocol/Internet Protocol (TCP/IP) came into use in the NCSs. However, the use of a LAN in NCSs poses several technical challenges including dealing with network latencies. To understand the latency issues, let us consider a simple client-server communication on an Ethernet as shown in Fig. 1. 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. 1.

This material is based upon work supported by the National Science Foundation under Grant No. CMS-0116642.

A. Ambike is currently a Software Engineer at NuView, Inc. in Houston, TX, USA (e-mail: ajitambike@yahoo.com).

W.-J. Kim is with Department of Mechanical Engineering, Texas A&M University, College Station, TX 77843-3123, USA (e-mail: wjkim@mengr.tamu.edu). Phone: (979)845-3645, Fax: (979)862-3989.

K. Ji is with Department of Mechanical Engineering, Texas A&M University, College Station, TX 77843-3123, USA (e-mail: kunji@tamu.edu).

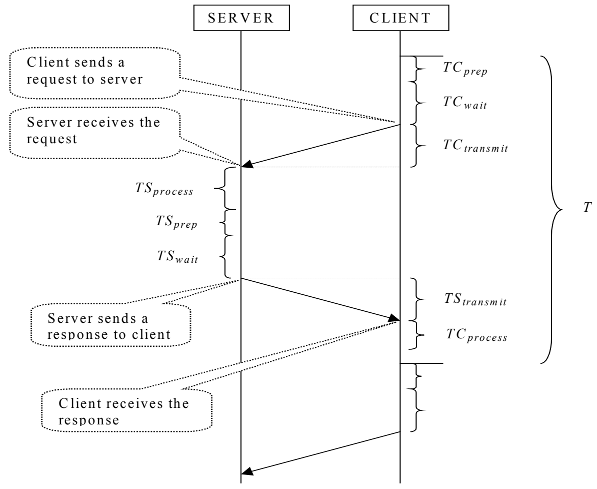

SERVER
CLIENT
Client sends a request to server
Server receives the request
TC_{prep}
TC_{wait}
$TC_{transmit}$
$TS_{process}$
TS_{prep}
TS_{wait}
T
Server sends a response to client
$TS_{transmit}$
$TC_{process}$
Client receives the response

Fig. 1. Time-delay components of the latency in a periodic client-server communication process.

TABLE I. NOMENCLATURE FOR 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

As shown in Fig. 1, a typical client-server communication in an NCS is periodic with a period *T*. The total communication process is required to be completed in one period. The total communication time or the latency is

$$\begin{aligned} T_{total} \;=\; & TC_{prep} + TC_{wait} + TC_{transmit} + TS_{process} \\ & + TS_{prep} + TS_{wait} + TS_{transmit} + TC_{process}\,. \end{aligned} \tag{1}$$

We apply a client-server architecture to a closed-loop NCS with the architecture shown in Fig. 2. The client side of the architecture hosts the test bed. The server side implements the controller. The request message sent by the client to the server carries the sensor data and the response message sent by the server to the client carries the control data.

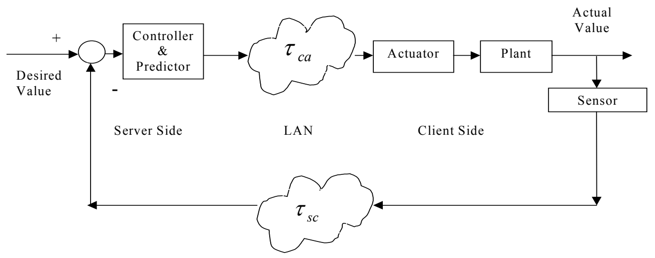


Fig. 2. Block diagram of a typical closed-loop NCS.

The creation of the message to be sent to the server is a time-driven process. Taking 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. For this kind of closed-loop NCSs to remain stable, these events have certain deadlines. If some of these deadlines would be missed, the stability of the NCS could be negatively affected. In the current research, the maglev test bed to be described in Section IV is open loop unstable. Srivastava [10] demonstrated that all the calculations in the feedback control loop should be completed in 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 happen 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 [11].

III. SELECTION OF REAL-TIME OPERATING ENVIRONMENT

A. Factors Affecting the Selection of Operating Environment

An appropriate OS is required to successfully implement the control architecture. Following factors were considered in the OS selection in the context of our maglev test bed.

1) Periodic tasks: The OS should allow execution of periodic tasks.

2) Time resolution: The time required for closing the control loop was expected to be not more than 1.4 ms. Thus, it was desirable that the time resolution be as small as 14 μs, which is 1% of the time required for closing the control loop.

3) Threads: There was also a need for multi-threaded programming to implementing the algorithms for closed-loop real-time control over the network.

Commercially available OSs such as Windows 2000, various versions of Unix, and Linux are not real-time OSs, and their performance with reference to above-mentioned factors turned out not to be very satisfactory. Two timing tests, as stated by Volz [11] were performed to observe their performances. The smallest amount of time that can be precisely measured on an OS is known as its clock resolution. The time required reading a clock is typically much less than its resolution. For example, the time required to access the clock in Redhat Linux 7.3 was found to be approximately 0.01 μs whereas its clock resolution was found to be 10 μs. Because the time for clock read is less than the resolution, many consecutive clock reads can be made before the value returned by the clock changes. The number of times the clock was accessed before the change in the value returned by the clock access function was recorded and then plotted. Fig. 3 represents the results of the first test on Windows 2000 and Redhat Linux 7.3 OSs. Significant variations in these plots indicate some other OS activities that are not deterministic.

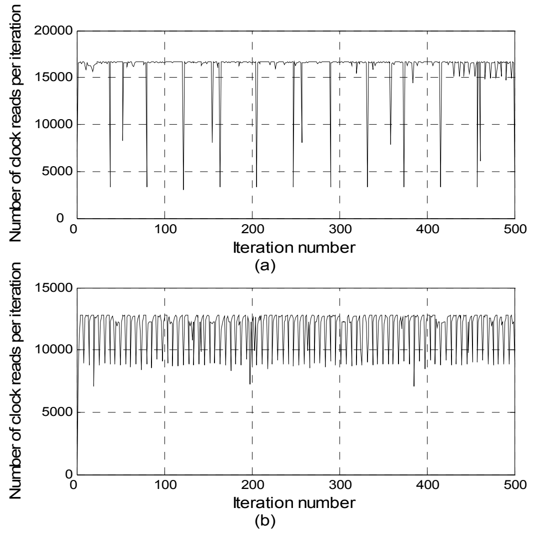


Fig. 3. Plots of the number of clock reads per iteration for the first timing test on (a) Windows 2000 and (b) Redhat Linux 7.3.

In the second test, the clock resolution was calculated and plotted over several iterations. There were variations in the values returned as clock resolution. Thus, the maximum value returned is considered the clock resolution for the corresponding OS. Fig. 4 presents the results of the second test on Windows 2000 and Redhat Linux 7.3 OSs, respectively. These simple tests demonstrated the non-real-time characteristics of the two popular OSs.

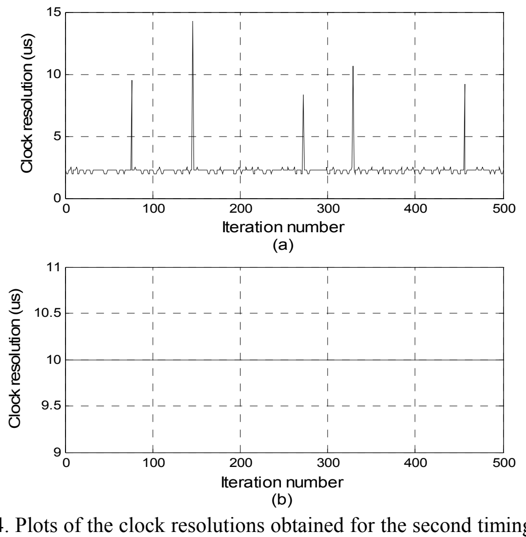


Fig. 4. Plots of the clock resolutions obtained for the second timing test on (a) Windows 2000 and (b) Redhat Linux 7.3.

We developed a novel real-time operating environment with RTAI with Linux that provided a competitive solution to the problem. Fig. 5 shows the results of first and second tests run on RTAI 24.1.12 with Redhat Linux 7.3.

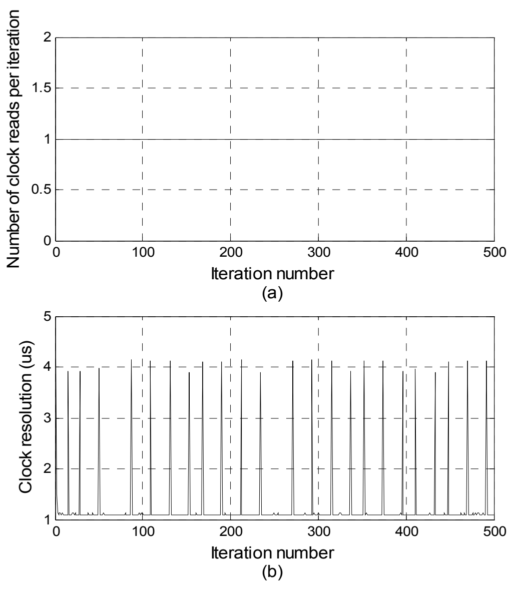


Fig. 5. Plots showing results of timing tests on RTAI 24.1.12 with Redhat Linux 7.3. (a) Number of clock reads per iteration for the first timing test. (b) Clock resolutions obtained for the second timing test.

In Fig. 5 (a), the straight line denotes that there was no significant non-deterministic OS activity. Although the spikes in Fig 5 (b) denote the variation in the clock resolution from 1 μs to 4.1 μs, the clock resolution reported is consistently less than 5 μs.

B. *Real-Time Application Interface*

The RTAI [12] was developed as a real-time operating environment solution at Dipartimento di Ingeneria Aerospaziale Politecnico di Milano (DIAPM). Based on the Linux kernel, it provides the ability to make Linux fully preemptive. Linux alone as an OS lacks real-time support. It is necessary to make some changes in its kernel behavior such as interrupt handling and scheduling policies to make it a real-time platform with low latency and high predictability of timing performances. 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. Compared to the commercially available real-time OSs, RTAI's performance is very competitive [12]. Table II summarized the typical performance of RTAI. Last but not the least, RTAI is open source, and free under the terms of the GNU Lesser General Public License.

TABLE II. RTAI'S TYPICAL PERFORMANCE.

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

IV. Development of the NCS Architecture

A. Test Bed

The single-actuator ball magnetic levitation system developed by Stephen C. Paschall, II as his senior honors project [13] was used as the test bed for this research. As shown in Fig. 6, an electromagnet is used to produce the force to support the ball against gravity. The force is produced by controlling the currents in the coil of the electromagnet using a control system implemented on a personal computer (PC). The digital controller designed by A. Srivastava [10] to stabilize this maglev system is given by

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

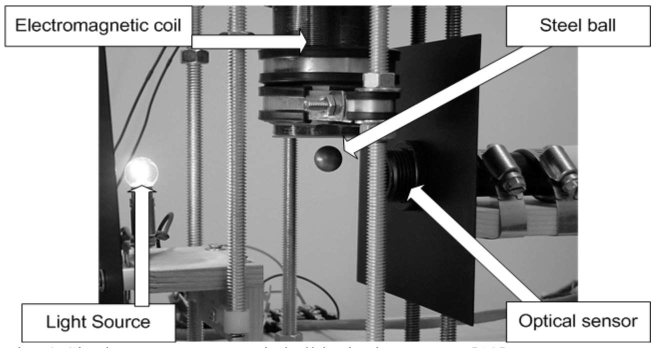


Fig. 6. Single-actuator magnetic ball levitation system [13].

B. Components and Protocols

The distributed NCS as shown in Fig. 7 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 was used for data acquisition and acts as the Client PC. A 100-Mbps Ethernet LAN was used as the medium of communication. This arrangement has the advantage of using standard commercial hardware. The network communication was based on the TCP/IP protocol suite and the programs were developed in the C programming language. These programs are available in Ajit Ambike's thesis [14]. 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 like CPU time and memory and introduces significant time delays in the communication [15]. 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 less network delays as compared to the TCP. Reduced round-trip time delays justified the use of the UDP for NCS applications.

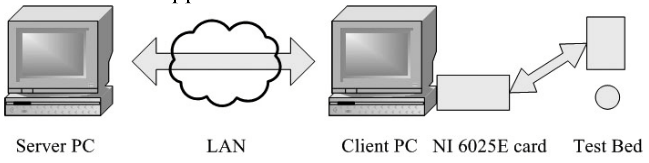


Fig. 7. Block diagram of the developed distributed NCS.

C. Timing of Events

In the past, synchronization of clocks was tried to coordinate the events in the networked control loop [16]. As it is a complicated process generating additional network traffic to deliver the synchronized clock signals, a comparatively simpler approach was developed in this paper. Our networked control architecture is based on a combination of time-driven and event-driven processes. Fig. 8 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* denote the control signal data transferred from the server to the client. The subscripts of these labels denote the sampling-period index (–7, …, 7) and indicate if the data is an estimate (*e*). For example, y_2 denotes the sensor data of the second sampling period, u_3 denotes the control signal data for the third sampling period and u_{2e} denotes the control signal estimate for the second sampling period.

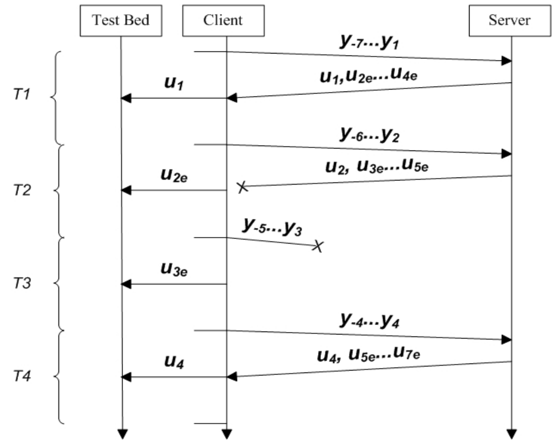
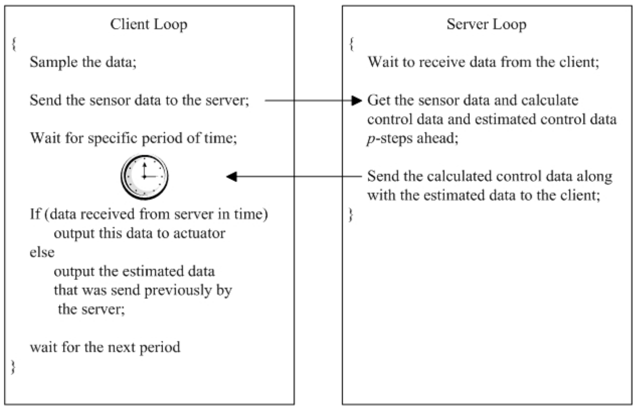


Fig. 8. Example timing diagram of the NCS communication. An arrow with a cross tip denotes that the data packet was lost. In the current architecture, lost data are not worse than late arriving data.

Fig. 9 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 [10]. Thus, for 333.33 Hz sampling frequency, the sampling and the actuation are offset by 1.4 ms on the client side. 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.



```
Client Loop
{
    Sample the data;

    Send the sensor data to the server;

    Wait for specific period of time;

    If (data received from server in time)
        output this data to actuator
    else
        output the estimated data
        that was send previously by
        the server;

    wait for the next period
}
```

```
Server Loop
{
    Wait to receive data from the client;

    Get the sensor data and calculate
    control data and estimated control data
    p-steps ahead;

    Send the calculated control data along
    with the estimated data to the client;
}
```

Fig. 9. Psuedocode for the client-server communication.

The process on the server side waits for the arrival of sensor data from the client. As soon as the data arrives, appropriate control is calculated and is sent using a UDP socket. In addition to the current control data, predicted control data for the next p sampling periods are sent to the client. The criterion to select this p will be discussed in Section V. Parametric predictors are used to calculate these estimates. These predictors require past sensor data, which are transmitted to the server along with the current sensor data.

The client stores and updates these values of predicted control signal after each data arrival from the server. In the event of time delays and data losses, these values of estimates 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]. But, with an open-loop unstable plant like our current maglev test bed, the stability of the system will be eventually lost in the event of excessive time delays. Thus multiple-step-ahead prediction is used to maintain system stability.

D. Predictor-Based Control Strategy

It was shown in Fig. 8 that the server calculated 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. Predictors were designed using MATLAB's system identification toolbox [17] and were based on an auto-regressive (AR) model [18]. A tradeoff exists between the accuracy of a predictor and the number of computations done by the predictor for each prediction. Generally higher-order predictors have a better accuracy of prediction than the lower-order ones of the same type. On the other hand, higher-order predictors take more computational time for a prediction than the lower-order ones of the same type. Based on these considerations, an 8th-order AR model was selected to design the predictor after numerous design iterations. Predictors were designed for up to 4-step-ahead predictions of sensor output. The 4-step-ahead prediction for control signal was calculated using the predicted sensor output data.

V. Experimental Verification of the NCS and its Real-Time Operating Environment

To verify the working of the networked control strategy, two sets of experiments were conducted. In the first set of experiments, the NCS architecture was tested for the maximum number of allowable consecutive delays. In the first experiment of this set, a data packet was simulated 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, the system lost stability as expected and the levitated ball could not maintain its equilibrium position. The output of the system is shown in Fig. 10 (a).

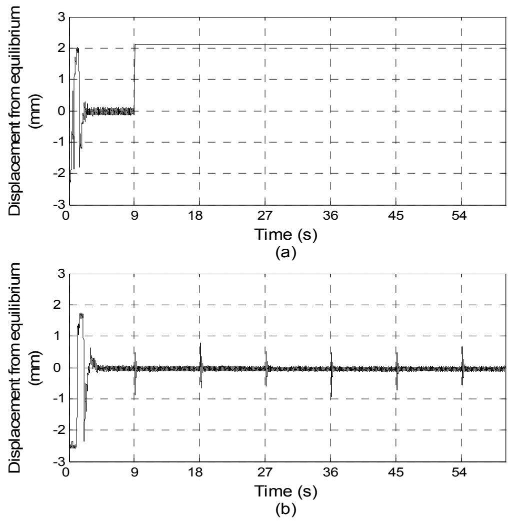


Fig. 10. Plot of displacement of ball from the equilibrium position vs. time. (a) System response for the first experiment of the first set. (b) System response for the second experiment of the first set.

In the second experiment of this set, the control signals based on predictions of the sensor data were used. Consecutive 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 experiment starting from zero. The maximum value of p was found to be two. In other words, the system accommodated two consecutive packet drops without loosing stability. The response of the system is shown in Fig. 10 (b). Every time these consecutive delays occurred, the system performance was degraded.

In the first experiment of the second set of experiments, an artificial delay of 2 ms was introduced with real-time-sleep function on the server side at the 7000th sampling period (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. 11 (a).

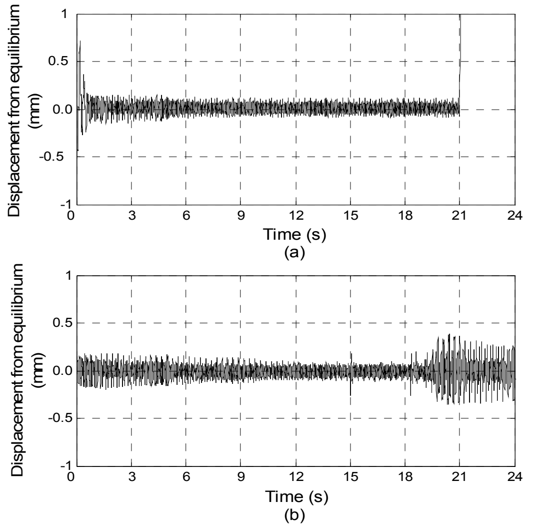


Fig. 11. Plot of displacement of ball from the equilibrium position vs. time. (a) System response for the first experiment of the second set. (b) System response for the second experiment of the second set.

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 6500th sampling period (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 simulation of one long time delay of sporadic nature in five consecutive delays. Thus, system stability was achieved in the events of up to 20% data loss in communication. The response of the system for q equal to five is shown in Fig. 11 (b). In the event of 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 was due to the use of the control based upon the estimated sensor data because actual sensor data were not available due to data-packet losses.

VI. Conclusions

A real-time operating environment is essential to properly time the key communication events in an NCS and to have a complete control on their execution. In this paper, we established a novel real-time environment for NCSs with RTAI 24.1.12 with Redhat Linux 7.3. The closed-loop control of an open-loop unstable magnetic ball levitation system was achieved over the Ethernet for its experimental verification. Due to its better real-time performance, the UDP was used in the communication architecture. A multiple-step-ahead predictor was implemented wherein predicted control signals were used to stabilize the system in the events of time delays and packet losses. The number of consecutive network delays compensated for by the system depends on the accuracy of the predictors. The predictors designed using an AR model were able to stabilize the maglev system for up to 20% data-packet losses in the communication network.

References

[1] H. Hu, L. Yu, P. W. Tsui, and Q. Zhou, "Internet-based robotic systems for teleoperation," *International Journal of Assembly Automation,*, 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. of the IEEE International Conference on Robotics and Automation,* San Francisco, CA, vol. 2, pp. 1607–1614, Apr. 2000.

[3] R. C. Luo, J. H. Tzou, and Y. C. Chang, "Desktop rapid prototyping system with supervisory control and monitoring through Internet," *IEEE/ASME Transactions on Mechatronic* , 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 Engineering Practice*, vol. 11, no. 7, pp. 805–817, July 2003.

[5] W. R. Ferrell, "Delayed force feedback," *IEEE Transactions on Human Factors in Electronics*, vol. 8, pp. 449–455, Oct. 1967.

[6] R. J. Anderson and M. W. Spong, "Bilateral control of teleoperators with time delay," *IEEE Transactions on Automatic Control*, vol. 34, no. 5, pp. 494–501, May 1989.

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

[8] J. Eker and A. Cervin, "Distributed wireless control using Bluetooth," in *Proc. of IFAC Conference on New Technologies for Computer Control*, Hong Kong, P.R. China, Nov. 2001.

[9] 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. 58–71, 2004.

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

[11] R. A. Volz, *Real-Time Computing,* Lecture Notes for CPSC 456, Department of Computer Science, Texas A&M University College Station, TX, 2003.

[12] P. Mantegazza, "DIAPM RTAI - real-time application," [Online]. Available: http://www.rtai.org.

[13] 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.

[14] A. Ambike, *Closed-Loop Real-Time Control on Distributed Networks*, Masters Thesis, Texas A&M University, College Station, TX, Aug. 2004.

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

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

[17] L. Ljung, "The system identification toolbox," [Online]. Available: http://www.mathworks.com/access/helpdesk/help/toolbox/ident.

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