Implementation of Model Based Networked Predictive Control System

Ahmet Onat
Sabanci University Fac. Eng & Natural Sciences
Orhanli, Tuzla 34956 Istanbul Turkey
onat@sabanciuniv.edu

M. Emrah Parlakay
Alcatel-Lucent
Burgemeester Elsenlaan 170, 2288 BH Rijswijk-ZH, Postbus 3292 The Netherlands
emrah.parlakay@alcatel-lucent.com.tr

Abstract

Networked control systems are made up of several computer nodes communicating over a communication channel, cooperating to control a plant. The stability of the plant depends on the end to end delay from sensor to the actuator. Although computational delays within the computer nodes can be made bounded, delays through the communication network are generally unpredictable. A method which aims to protect the stability of the plant under communication delays and data loss, Model Based Predictive Networked Control System (MBPNCS), has previously been proposed by the authors. This paper aims to demonstrate the implementation of this type of networked control system on a non-real-time communication network; Ethernet.

In this paper, we first briefly describe the MBPNCS method, then discuss the implementation, detailing the properties of the operating system, communications and hardware, and later give the results on the performance of the Model Based Predictive Networked Control System implementation controlling a DC motor. [1]

1 Introduction

A Networked Control System (NCS) is a feedback control system where the control loop is closed over a communication network consisting of actuators, sensors and controllers, each of which are computer nodes on the network. Actuators and sensors generally also have some computational capability. This distributed structure is advantageous because of its inherent flexibility, reconfigurability and reduced vulnerability to noise and calibration errors.

In a NCS, sensor nodes have the task of measuring one or multiple plant outputs and transmitting the measured values over the network. Actuator nodes have the task of applying commanded values received over the network to the plant by means of suitable actuators. Controller nodes use the plant outputs that they receive from sensor nodes to calculate control outputs by a control algorithm and send them to the actuator nodes to be applied to the plant. The data that travels over the network is encapsulated in packets.

The complexity of design and the communication delays are drawbacks of NCS's. With the addition of a communication network in the feedback control loop, the complexity of analysis and design for a NCS increases because delay in the control loop has to be accounted for. There are essentially three kinds of delays in a NCS which are dependent on the network scheduling policy and are generally not constant or bounded in common network protocols: Communication delay between the sensor node and the controller node that has occurred during sampling instant t_k : $\tau_{sc}(t_k)$, computation delay in the

[1]This work was supported in part by the Scientific and Technological Research Council of Turkey, project code 106E155.

controller node that has occurred during sampling instant $t_k : \tau_c(t_k)$, and communication delay between the controller node and the actuator node that has occurred during sampling instant $t_k : \tau_{ca}(t_k)$. The length of the transfer delay depends on network load, priorities of the ongoing communications, electrical disturbances and various other factors. Sensor and actuator nodes also have some computational load and therefore some delays that can be expressed respectively as $\tau_s(t_k)$ and $\tau_a(t_k)$, but these delays can be considered as fixed and the sensor node calculation delay can be included in $\tau_{sc}(t_k)$ and the computation delay at the actuator node can be included in $\tau_{ca}(t_k)$. The total delay from sensing to actuation is the sum of the above delays:

$$\tau(t_k) = \tau_{sc}(t_k) + \tau_c(t_k) + \tau_{ca}(t_k) \qquad (1)$$

The computational delay τ_c is variable, not only because of the time the control algorithm takes, but because of the scheduling algorithm used [1].

2 Background

Digital control theory has developed over the years mainly assuming fixed response time to the plant, and mainly focused on the control algorithm whereas real-time systems development has largely avoided the functional requirements and concentrated on the timing requirements for implementation. NCS can be considered as a step in combining the two approaches, considering both the functional and timing requirements of a control system. Several methods have previously been proposed to improve stability of NCS [1, 2, 3, 4]. Dead bands proposed by Otanez and Moyne [5] aim to reduce the amount of communication by eliminating repetitive transmissions of similar data, thus improving network conditions. A similar idea is also considered in [6, 7, 8]. However the network is assumed to be loss-less. Gain adaptation by Mo-Yuen and Tipsuwan [9] and network observers by Natori and Ohnishi [10] observe the condition of the network and compensate for the effect of delay in the control algorithm by adjusting the gain or adding a negative feedback term. However they consider the changes in network to be relatively slow or the network delay times to be symmetric (sensor node to controller node delay is same as controller node to actuator node delay). Some a-priori knowledge of the delay is assumed.

General predictive control methodology is relevant to NCS's [11, 12, 13] and Model predictive controllers are used in similar scenarios as in [14, 15] but they either do not take into account the synchronization between the nodes or they are not set up to be networked control systems, because they rely on a direct link between the sensor node and controller node. This means that both controller and sensor tasks reside within the same node of the network or they do not communicate at all over a network. The reason is that if the sensor node to controller node transmission fails then the basis for predictions is lost. Also a-priori knowledge of the reference signal is assumed in the model predictive control. The MBPNCS method proposed by the authors is currently being applied to this area to adapt it to network control.

Addressing these problems, MBPNCS method improves the performance of a basic NCS under variable time delays and packet loss. Standard NCS architecture is assumed and no direct links are required, therefore the method can be applied to existing NCS's. A-priori knowledge of the reference signal is not assumed as this is not possible with most systems.

3 Model Based Predictive Networked Control Systems

MBPNCS improves the robustness and stability of networked control systems to data loss by holding a model of the plant within the controller and calculating the current and predicted control output to the plant for several time steps into the future at every sampling instant. All of these outputs are then sent to the actuator node at once. If there was no data loss in the controller to actuator link, the actuator applies the first control output to the plant. In case of data loss, a previously sent prediction is applied to the plant at each successive sampling instant, hence the name model based predictive networked control system.

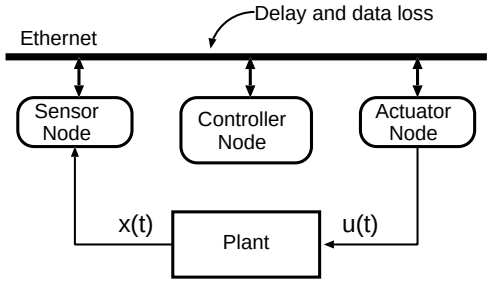

FIGURE 1: *Model based predictive networked control system*

The MBPNCS control system is composed of five parts: a communication network, where we assume that packet loss and delay occurs completely randomly (despite the fact that noise in networks is generally more correlated, a complete random behavior was preferred for simulations for simplicity), one sensor node, one controller node and one actuator node,

and finally a model of the plant $\hat{P}$ residing inside the controller node.

The sensor node samples the plant states $x(t_k)$ with period h and sends them to the controller at every sampling instant t_k. The rate terms of the control algorithm pertaining to the plant states are calculated at the sensor node since continuity of plant states cannot be assured at the controller node because of interruption of communication. The plant states and the rate terms are encapsulated in a network packet and sent to the controller node.

The controller node not only calculates the control output for the current plant state $u(t_k, 0)$ (notation will be clarified shortly) using a control algorithm, but also uses the plant states just received $x(t_k)$, to initialize the internal model and through an iterative process, generates a series of future control outputs, $\hat{u}(t_k, i)$ and state estimates $\hat{x}(t_{k+i}|t_k)$ where $i; i = 1, 2, ..n$ is the index of control output signal estimate that is expected to be applied at time $t_k + ih$ in the future if communications were to fail. If $x(t_k)$ was not available at the time of computation because of data loss or communication delay, its estimate $\hat{x}(t_k|t_{k-1})$ obtained using the model $\hat{P}$ from $x(t_{k-1})$ or if that is not available, $\hat{x}(t_{k-1}|.)$ from previous estimates. For this estimation to be valid, it must be assumed that the previous control output was transmitted properly and applied to the plant. How this restriction can be relaxed will be addressed shortly.

The fact that a state estimate has been used is important along the control decision line. Therefore this information is stored in a *sensor flag* (SF), which is set to high if current control output is based on actual plant sates from the sensor node, and low if state estimates were used. The control outputs are collected in a control packet $Pt(t_k)$ consisting of $n + 1$ control outputs and a sensor flag: $[u(t_k, 0), \hat{u}(t_k, 1), \cdots, \hat{u}(t_k, n), SF]$.

The number of predictions n is chosen based on factors such as the accuracy of the plant model $\hat{P}$, the packet size compared to the network bandwidth, and available processing power.

Finally the actuator node receives packet $Pt(t_k)$ and applies the control output $u(t_k)$ to the plant at every sampling instant. If a new packet does not arrive on time because of data loss or communication delay, a predicted control output $\hat{u}(t_{k-i}, i)$ received previously is applied. This could cause the above mentioned problem of predicted plant states deviating from the actual plant states since if a sensor to controller communication loss also occurred simultaneously, the controller assumes that $u(t_k)$ was last applied to the plant when calculating its state predictions. In our method, this is called the loss of synchronization (slightly abusing the term), and our method is designed so that the actuator node is responsible for coping with the situation.

The synchronization or loss thereof is conveyed to the actuator node using the SF flag in the control packet and the information of actual packet loss. The actuator node has two modes, the synchronized mode and the interrupted mode.
The synchronized mode indicates that the states of the plant model are synchronized with the plant states. If the actuator node receives a control packet from the controller node when it is in the synchronized mode then it applies the first control output from that packet to the plant, which would be $u(t_k, 0)$. If a transition of SF from high to low occurs in consecutive control packets indicating that the controller is not receiving actual plant states, but there is no controller to actuator data loss, then the actuator keeps applying the first control output from the received packets $u(t_{k+j}, 0)$, since this does not violate the assumption made by the controller that these control outputs are being applied to the plant, and the actuator node stays in synchronized mode. If data is lost due to network delay or packet loss, the actuator node enters the interrupted mode.
In the interrupted mode, the actuator applies $\hat{u}(t_k, i)$ of the last control packet received in synchronized mode at every sampling instant t_{k+i}, $i = 1, 2, \cdots, n$ until the last sample is reached or communication is restored. However, if the first control packet received in this mode has a low SF indicating that the controller is using state estimates based on the wrong assumption of applied control signal, then the packet is rejected. If the last prediction $\hat{u}(t_k, n)$ is reached without the communication being restored, the output is kept constant at that value thereafter. The actuator node enters synchronized mode when a control packet with a high SF is received. This state machine behavior is shown in Figure 2.

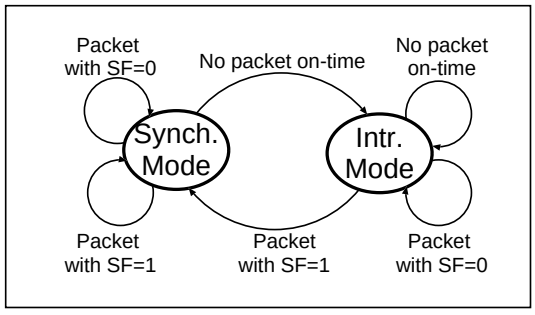

FIGURE 2: *State transition diagram for actuator modes*

All computer nodes run periodic tasks as a computational model. Packet loss between the sensor node and the actuator node is compensated at the controller node by prediction and packet loss between

the controller node and the actuator node is compensated at the actuator node by usage of the selection algorithm explained above and predicted control outputs. Late arriving packets are discarded in this work. A time synchronizing method is assumed to be used among the computer nodes. This is not a strong assumption because the network is generally physically small and the amount of synchronization accuracy is comparable to the sampling time.

4 Real-Time Linux implementation of MBPNCS

There are two aspects of the implementation; (hard)real-time computing and soft real-time communication. The timing requirements for computing were realized by using a small Linux distribution with 2.4.20 series kernel using RT-Linux patch V3.2-pre3. As run-time environment Clash V1.2 file system based on compact flash sotage medium(CF) [16] and Busybox V1.00 combined executive were used [17]. The whole system was designed to run from RAM disk to minimize run-time delays caused by disk access and eliminate the burnout problem of the Flash media due to repetitive writes. The resulting RT Linux distribution was named TuxSA and has been in service since 2003. Work is underway to upgrade to a more common distribution and new RT Linux version. As a hardware platform, Advantech PCM3370 PC104 boards with NS Geode processor running at 300MHz equipped with 128MB RAM and 64MB CF flash disks were employed. For connecting sensors and actuators Kontron ADIO 128 12 bit analog digital and digital analog converter boards and MSI-P400 incremental encoder interface boards from Microsystems Technology were used. Other electronic circuits such as motor driver, and encoder adapter were also built so that it is easier to identify their parameters and incorporate them in the plant model. Although the processor is not suitable for hard real-time processing with low jitter, control loop times of $100\mu s$ were accomplished with the overlaying Linux kernel shut down. Details of how Real-Time Linux kernel is accommodated and its relationship with the non-real-time Linux kernel running atop is not discussed here, but can be found in the literature such as in [18].

Since the implementation runs in kernel space as kernel modules, low level code was written to directly access the sensor interface boards from scratch. This has the benefit of removing extra layers of software that can cause unpredictable delays, however at the same time it makes hardware updates difficult to perform mainly because hardware access methods are usually not disclosed by the manufacturers.

MBPNCS method was implemented on the sensor, controller and actuator nodes as simple RT-Linux periodic tasks that run with a period of $1ms$. The code was simply copied over from the simulation of the system in TrueTime as explained in Section 5.1, with minor modifications such as placing in empty periodic task shells and using the appropriate hardware access and communication routines as described above. There was no clock synchronization used, mainly because the nodes were synchronized at the initialization by the signal from the sensor node and since the run-times were in the order of several tens of minutes to a few hours, there was no noticeable clock skew between the nodes. For actual implementations that are required to run for indefinite amounts of time, an active clock synchronization method should be employed.

The second main component of the system is communication. Although hard real-time communication is not required (and actually opposes the purpose of the study), at least soft real-time network communications is desired so that the actual performance can be degraded in a controlled manner for experiments. Since no acknowledge or automatic resend is used in MBPNCS, user datagramm protocol (UDP) was used. To allow for soft real-time performance, real-time socket package for RT Linux patched kernel, *rtsock* was used. Since this package is not well supported, incorporating it has been problematic. Another option is to use the non-real-time sockets provided by Linux, and a real-time FIFO buffer between them. To connect the nodes together a closed network segment was built using a hub which is non-switching to allow for collisions. Since there is no other traffic on the network and the existing traffic is phased suitably, the actual rate of collisions is negligible. Packet loss was accomplished by a pseudorandom routine rejecting packages, as explained in Section 5.3. In an actual implementation, the hub can be replaced with a switching one to improve performance.

Finally, a word about storing the data generated by experiments should be said. Since the TuxSA distribution is completely RAM disk based, there are a few options. One of them is to re-mount the boot CF disk, and write the data on a suitable folder there to be manually retrieved later. Another method is to NFS mount a partition on a server to store data there. However, since NFS support was not available in TuxSA distribution, this was not possible. Also possible is to record the data on a file on RAM disk, and retrieve it after stopping the simulations. This was the preferred method for this study.

5 Results

The MBPNCS method was tested using computer simulations using the TrueTime toolbox of Matlab [19, 20] and experiments were performed with industrial computers and Ethernet communication network.

5.1 Simulation Environment

TrueTime toolbox is designed to simulate real-time computer networks at a low level of abstraction where it simulates computer systems at instruction execution level and communication network at data transport level. Therefore, we can say our results are close to actual implementation. Implementation of the hardware is detailed in Section 4.

A DC servo motor described by the following transfer function is used as the system plant [21]:

$$G(s) = \frac{1000}{s(s+1)} \tag{2}$$

A PD controller is implemented according to the following equations;

$$KP(t_k) = K(r(t_k) - y(t_k)) \tag{3}$$

$$KD(t_k) = \alpha_d KD(t_k - 1) + \beta_d(y(t_k - 1) - y(t_k)) \tag{4}$$

$$\alpha_d = \frac{T_d}{N_h + T_d} \tag{5}$$

$$\beta_d = \frac{NKT_d}{N_h + T_d} \tag{6}$$

$$u(t_k) = KP(t_k) + KD(t_k) \tag{7}$$

where $r(t_k)$, $y(t_k)$, $u(t_k)$ are reference, plant output, control output and $KP(t_k)$, $KD(t_k)$ are proportional and derivative components of control output, K is the proportional gain and t_k, is the sampling instant, N, N_h, α_d and β_d are constants. The value $y(t_k)$ is obtained by $H * x(t_k)$ where H is the output matrix of the plant and $x(t_k)$ are the plant states at time t_k. The performance of MBPNCS method is compared with a basic Networked Control System (bNCS) where only the sensor node runs a periodic task and the controller and actuator nodes run event driven tasks that function only when they receive a message from the network to calculate control output and apply it to the plant respectively. As performance metric the root mean square of the error between the reference and plant output is used.

5.2 Simulation Results

The sampling time of the system is $0.001s$, and there is a phase delay of $0.0001s$ between sensor, controller and actuator node periods to ensure that the network has time to deliver the data packets between the nodes. Such a phasing was not used for the experiments. Simulations are made with a perfect model of the plant to prove that the concept is functional. Further simulations examining imperfect models will be performed in the future.

Under ideal network condition of no packet loss both the MBPNCS and bNCS display identical results, Figures 3 and 4.

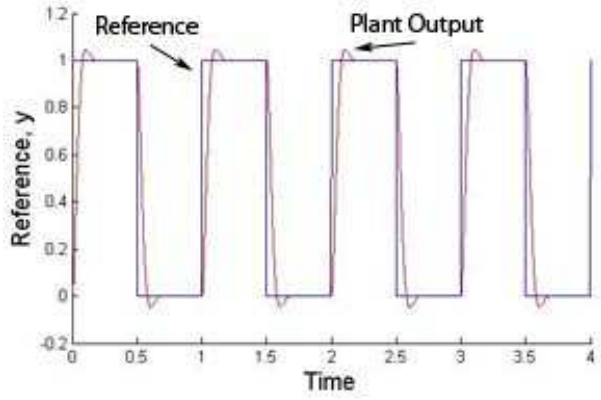

FIGURE 3: *Basic NCS, ideal conditions RMS error: 0.2324*

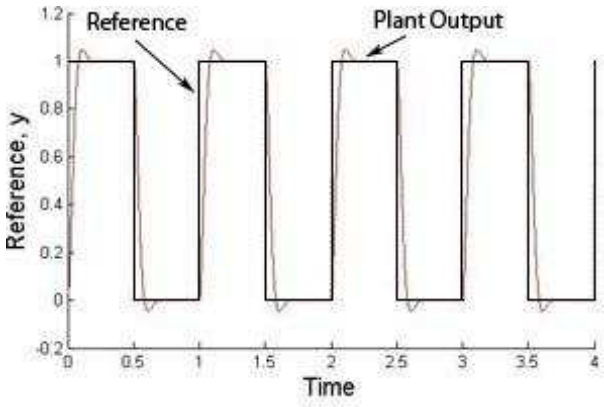

FIGURE 4: *MBPNCS, ideal conditions RMS error: 0.23252*

As packet loss increases, degradation in performance is observable with the increasing RMS errors. However the reason for the degradation in control quality is different in both systems. The cause of increased RMS error in the bNCS is loss of stability because of increasing loop delay. The bNCS system becomes unstable after around 20% of packets lost or delayed(Figs 5 and 6). On the other hand the increase in RMS of the MBPNCS is because even if packets are late, a calculated control output is applied to the plant. However after the calculation of this control output the reference may have changed.

Therefore the plant is controlled towards an old reference. The retardation of the reference can be seen clearly on Figure 8 where the reference and plant output are shown together with the control output signal estimate i(offset by -4 for clarity) used from the control packet. It can be seen that the plant is able to catch the reference once the actuator reenters a synchronized mode and the actuator node does not have to remain in synchronized mode to be able to go to the reference. Note that our system does not have a-priori knowledge of the reference signal in contrast to other research such as in [15]. MBPNCS remains stable, even for extreme rates of packet loss such as 90% (Fig. 8).

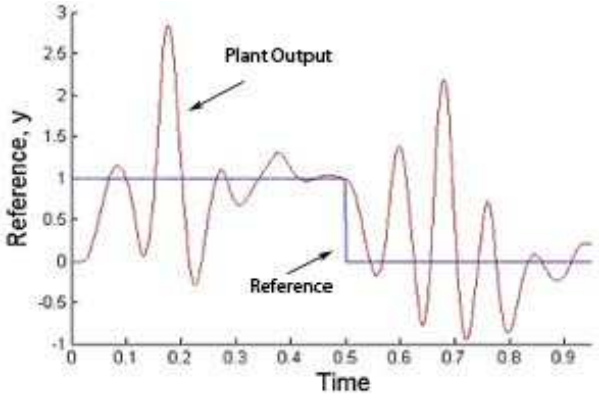

FIGURE 5: *Basic NCS %20 packet loss RMS error: 0.66509*

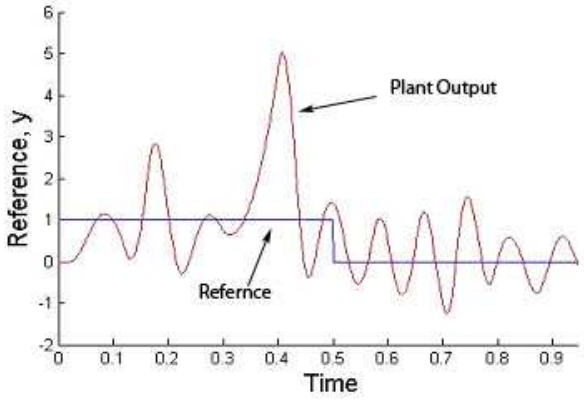

FIGURE 6: *Basic NCS %30 packet loss RMS error: 1.023*

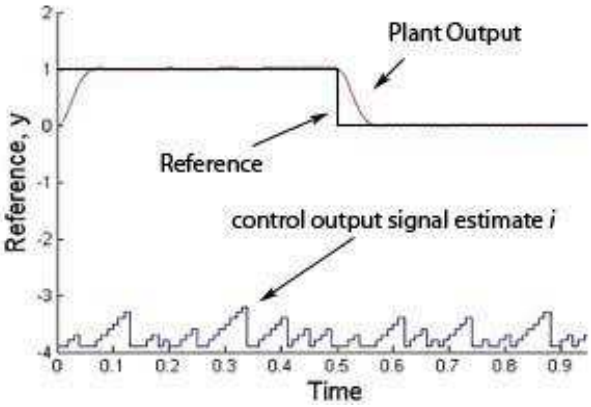

FIGURE 7: *MBPNCS %50 packet loss RMS error: 0.22644*

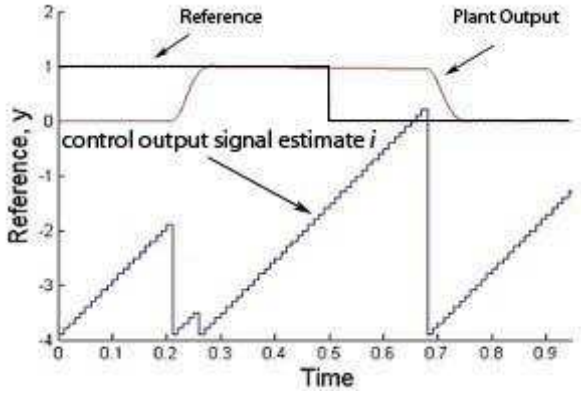

FIGURE 8: *MBPNCS %90 packet loss RMS error: 0.65153*

The performance of MBPNCS under sensor noise has been investigated in [22] and the effect of jitter in the system is analyzed in [23].

5.3 Experiment Results

A physical setup comprising of three computers, a dedicated Ethernet network, and a DC motor with encoder and drive electronics was prepared for experiments as explained in Section 4. The sampling time was set at $1ms$. Again speed control of DC motor using PID method was targeted. Motor parameters were measured and used as plant model.

Since Ethernet is stochastic, controlled amounts of delay and data loss on the network were implemented by randomly accepting or rejecting incoming data packets on the computer nodes at a specified rate.

Tests performed on the experimental platform are similar to those for the simulations. The first

test was for a practically no data loss or delay scenario of 0.33% for MBPNCS, to see performance under ideal conditions. The small amount of data loss is caused by the Ethernet hub. The results are given in Figure 9. The reference, plant output and number of consecutively missed packets can be seen in the figure. The number of consecutively missed packets has been offset to avoid cluttering. It can be seen that the system can control the speed with little error. The noise at the $50 rad/s$ reference level is also present if a completely centralized conventional controller is used and is believed to happen because of sensor noise which is not filtered.

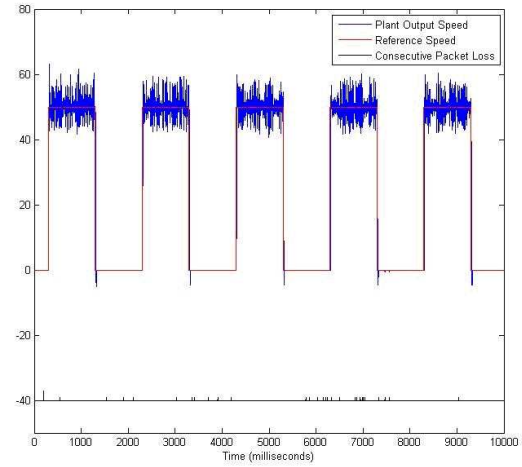

FIGURE 9: *MBPNCS experiment with 0.33% packet loss*

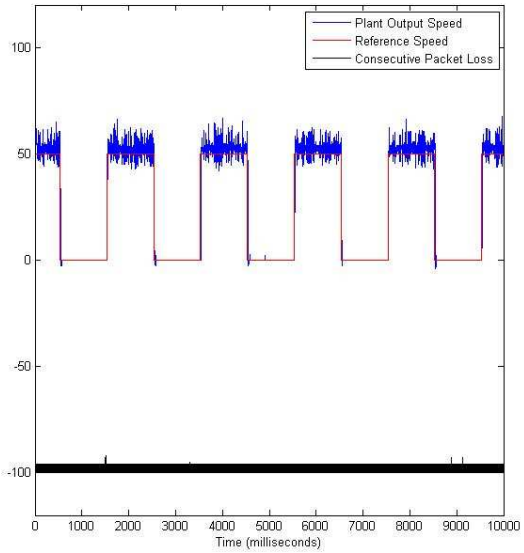

FIGURE 10: *MBPNCS experiment with 70% packet loss*

As data delay and loss rate over the network is increased, the MBPNCS method holds up well. As an intermediate value, Figure 10 shows the case where 70% of the packets are discarded for being late or dropped. Here, we assume a more realistic scenario where the network is completely down for the given rate, and functioning for the rest.

Finally, if the data delay and loss rate is increased further, system becomes unstable, as shown in Figure 11, at around 98% of the packets being delayed or lost. Performance begins to degrade around 90%, and deteriorates steadily.

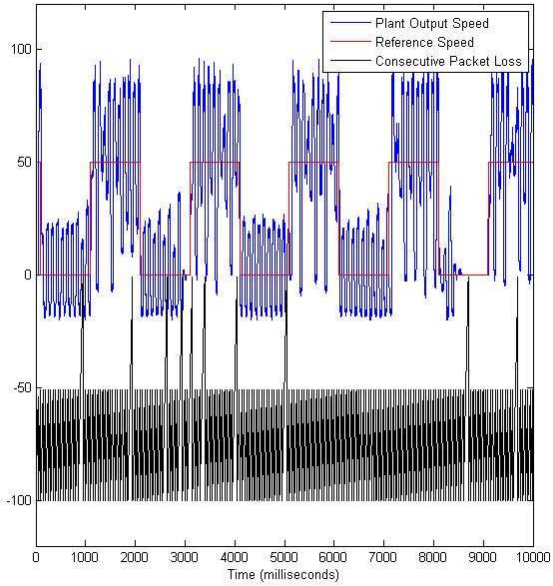

FIGURE 11: *MBPNCS experiment with 98% packet loss*

As a comparison, we also tested the case where the network is not completely down, but sensor to controller node packets and controller to actuator node packets are delayed or lost without any correlation. The result of that case can be seen in Figure 12 with performance similar to the case in Fig. 10

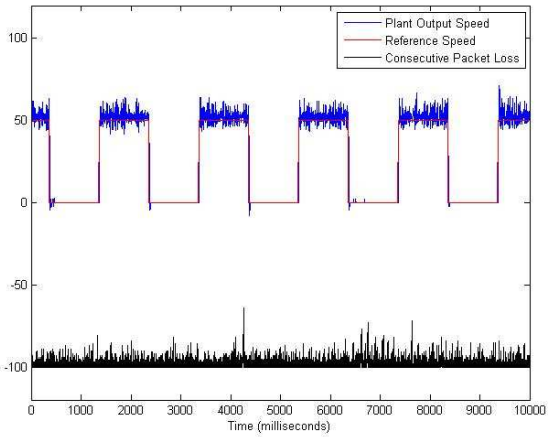

FIGURE 12: *MBPNCS experiment with 70% packet loss (packet loss not correlated).*

These results show that the performance in simulations and experiments are similar.

6 Conclusion

In this work, the implementation of model based predictive network control system method (MBPNCS) using Ethernet as a non real-time network is presented. Real-Time Linux is used to guarantee real-time performance of the computer nodes.

The method is applied both to a DC servo motor simulation and and experimental setup with real-time computers, to examine various aspects of the MBPNCS method. It has been observed that the method is robust against network packet loss. The destabilizing effect of packet loss is reduced to unresponsiveness to the reference command which is an inevitable consequence of communication loss.

An improved version of the implementation that incorporates more powerful computer nodes is now under construction to apply MBPNCS method to the inverted pendulum problem.

References

[1] M Toerngren, *Fundamentals of implementing real-time control applications in distributed computer systems*, Real-Time Systems, Vol.14, 219-250, 1998.

[2] M.S. Branicky and S.M. Phillips and Wei Zhang, *Scheduling and feedback co-design for networked control systems*, IEEE Conf. On Decision and Control, Vol.2, pp.1211-1217, 2002.

[3] M. S. Branicky and V. Liberatore and S. M. Phillips, *Networked control system co-simulation for co-design*, American Control Conf., Vol.4, pp.3341-3346, 2003.

[4] Q Ling and M. Lemmon, *Robust performance of soft real-time networked control systems with data dropouts*, IEEE Conf. On Decision and Control, 2002.

[5] P Otanez and J. Moyne and D. Tilbury, *Using deadbands to reduce communication in networked control systems*, American Control Conf., 2000.

[6] J. Yook and D. Tilbury and N. Soparkar, *Performance evaluation of distributed control systems with reduced communications*, IEEE Control Syst. Magazine, Vol.21, No.1, pp.84-99, 2001.

[7] J. K. Yook and D. M. Tilbury and H. S. Wong and N. R. Soparkar, *Trading computation for bandwidth: state estimators for reduced communication in distributed control systems*, Proc. JUSFA Japan-USA Symposium on Flexible Automation, 2000.

[8] J. Yook and D. Tilbury and K. Chervela and N. Soparkar, *Decentralized, Modular Real-Time Control for Machining Applications*, American Control Conf., pp.844-849, 1998.

[9] C. Mo-Yuen and Y. Tipsuwan, *Gain adaptation of networked DC motor controllers based on QoS variations*, IEEE Trans. on Indust. Electronics, Vol.50, No.5, pp.936-943, 2003.

[10] K. Natori and K. Onishi, *Time Delay Compensation by Communication Disturbance Observer in Bilateral Teleoperation Systems*, Adv. Motion Control, pp.218-223, 2006.

[11] D.W. Clarke and C. Mohtadi and P.S. Tuffs, *Generalized predictive control - Part I the basic algorithm*, Automatica, Vol.23, No.2, pp.137-148, 1987.

[12] D.W. Clarke and C. Mohtadi and P.S. Tuffs, *Generalized predictive control - Part II extensions and interpretations*, Automatica, Vol.23, No.2, pp.149-160, 1987.

[13] D.W. Clarke and C. Mohtadi, *Properties of generalized predictive control*, Automatica, No.6, Vol.25, pp.859-875, 1989.

[14] J.B. Rawlings, *Tutorial Overview of Model Predictive Control*, IEEE Control Systems Magazine, No.3, Vol.20, pp.38-52, 2000.

[15] G. P. Liu and J. X. Mu and D. Rees D, *Networked predictive control of systems with random communication delay*, UKACC Int. Conf. on Control, 2004.

[16] *http://sourceforge.net/projects/bclash*

[17] *"http://www.busybox.net"*

[18] *http://www.realtimelinuxfoundation.org*

[19] D. Henriksson and A. Cervin and K. Arzen, *TrueTime: Real-time Control System Simulation with MATLAB/Simulink*, Proc. of the Nordic MATLAB Conference, 2003.

[20] D. Henriksson and A.Cervin and K. Arzen, *Simulation of Control Loops Under Shared Computer Resources*, Proc. 15th IFAC World Congress on Automatic Control, 2002.

[21] K.J. Astrom and B. Wittenmark, *Computer Controlled Systems Theory and Design, 3rd ed.*, Prentice Hall, 1997.

[22] A. Onat and A.T. Naskali and E. Parlakay,*Model based predictive networked control systems*, "IFAC08", submitted.

[23] A.T. Naskali and A. Onat,*Jitter analysis of model based predictive networked control system*, 6th WSEAS Intl. Conf. on Applied Computer Science, Vol 1, pp.199-206, 2006.