IMPLEMENTATION OF A DISTRIBUTED CONTROL SYSTEM USING REAL-TIME OPERATING SYSTEM

By
MEHMET EMRAH PARLAKAY

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

SABANCI UNIVERSITY
Summer 2007

IMPLEMENTATION OF A DISTRIBUTED CONTROL SYSTEM
USING REAL-TIME OPERATING SYSTEM

APPROVED BY:

AHMET ONAT .....
(Dissertation Advisor)

GÜLLÜ KIZILTAŞ .....

HÜSNÜ YENİGÜN .....

AYHAN BOZKURT .....

VOLKAN PATOĞLU .....

DATE OF APPROVAL: .....

ABSTRACT

In a typical distributed control system, computer nodes communicate through a common communication channel that introduces data loss and random delays. Supplying a generic solution to these constraints is hard due to the complexity and large variety of possibilities that may affect these constraints in real life applications. In a modern communication network, if data is corrupted during transmission, it can be resent. However, it is not feasible to retransmit in control applications; if the packet contents correspond to measured plant outputs, then the most recent data should be measured and sent instead, or if the packet contents correspond to a control signal and the retransmission would cause the control signal to be applied late to the plant, it would be better to recalculate the signal and send it again.

This thesis is an attempt to implement a distributed control system design method, Model Based Predictive Networked Control System (MBPNCS), which accepts the fact that arbitrary delay and data loss may happen. The MBPNCS method approaches the problem by using a plant model to predict a predefined number of future states of the plant and respective control signal for each, to compensate for the possible delay and data loss that can take place during the communication between nodes.

In this work, after previous works have been examined, predictive control method that is used in the implementation is introduced. Design and implementation of the methodology is explained in detail and results of the tests are presented.

ÖZET

Tipik bir dağıtılmış kontrol sisteminde, ağ üstündeki kontrol sisteminin parçası olan cihazlar, veri kaybı veya rasgele gecikmelerin yaşanabileceği bir bilgisayar ağı kullanarak haberleşir. Bu problemlere genel bir çözüm sunmak, sistemin karmaşıklığından ve bu şartları etkileyebilecek muhtemel unsurların fazlalığından dolayı zordur. Modern bilgisayar ağlarında, bir veri kaybı yaşandığı zaman tekrar yollanarak problem giderilmektedir. Fakat gerçek zamanlı uygulamalarda gecikme veya kayıp halinde veriyi tekrar yollamanın bir anlamı yoktur. Eğer kaybolan gerçek zamanlı veri, ölçülen tesis çıktısı ise, tesis çıktısı tekrar ölçülüp en güncel haliyle yollanmalıdır. Eğer kaybolan gerçek zamanlı veri, kontrol çıktısı ise, çıktının tekrar gönderilmesi, çıktının tesise geç bir zamanda uygulanması demektir; bu durumda kontrol çıktısının tekrar hesaplanıp gönderilmesi daha sağlıklıdır.

Bu tez, ağ üzerinde rasgele veri kaybı veya gecikmenin olabileceğini kabullenen bir dağıtılmış kontrol sistem metodunu uygulamaya geçirmek adına yapılmış bir çalışmadır. Bu çalışmada kullanılan metot, ağ üzerinde olabilecek muhtemel veri kaybı ve gecikmeden kaynaklanabilecek problemleri, tesisin bir modelini çalıştırıp belirli sayıdaki sonraki tesis durumunu ve karşılık gelen kontrol çıktısını hesaplayarak telafi etmeye çalışmaktadır.

Bu çalışmada, bu konuda yapılmış daha önceki çalışmalar incelendikten sonra, uygulamaya geçirilen tahminsel kontrol metodu anlatılmıştır. Daha sonra tasarım ve uygulama aşamaları detaylarıyla anlatılmış ve test sonuçları yorumlanmıştır.

To my father who met his untimely death,
to my mother who insisted on me to go on
and to my brother, to my best friend who was
always there

ACKNOWLEDGEMENTS

I would like to, first of all, express my deepest gratitude to my thesis advisor, Assistant Prof. Dr. Ahmet Onat, who was always there for his invaluable help, encouragement and guidance for this work. For me, it was such a unique opportunity and, indeed, privilege to be able to work with him.

Including the undergraduate education, during my seven years at Sabancı University, I have been blessed with many good friends who have been with me all along the way. I will always be grateful to them for the times spent together.

It is hard to express my gratitude to Shahzad Khan, as a talented young scholar, dedicated his weeks for me and became literally a remote controller to make this thesis finished in time in a distributed work environment. This work would not be finished without his infinite patience and support. I also would like to thank him for enduring the "next year" and for functioning as a cook, alarm clock, tour guide and valuable friend whenever needed.

Special thanks to İlker Sevgen for his technical support, guidance and friendship.

Many thanks to Yasser El-kahlout, Meltem Elitaş, Emrah Deniz Kunt, Teoman Naskali (especially for his thesis work and related help), Selim Yannıer, Ahmet Fatih Tabak, Ertuğrul Çetinsoy, Elif Hocaoğlu, Şakir Kabadayı, Ahmet Altınışık and many other friends who are not mentioned here for their support and friendship.

Finally, I would like to thank to Prof. Dr. Asıf Şabanoviç, Assist. Prof. Dr. Kemalettin Erbatur, Assist. Prof. Dr. Hüsnü Yenigun, Assist. Prof. Dr. Ayhan Bozkurt, Assist. Prof. Dr. Güllü Kızıltaş Şendur and Assist. Prof. Dr. Volkan Patoğlu for their invaluable patience and guidance.

TABLE OF CONTENTS

LIST OF FIGURES

TABLE OF ABBREVIATIONS

DCS	Distributed Control System
MBPNCS	Model Based Predictive Networked Control System
A/D	Analog to Digital converter
D/A	Digital to Analog converter
IP	Internet Protocol
TCP	Transport Control Protocol
UDP	User datagram protocol
MSE	Mean square error
CPU	Central Processing Unit
GNU	GNU is Not Unix
RMS	Root mean square
NCS	Networked Control System

1 INTRODUCTION

The advantages of digital control systems have been long known and used for a long time. Digital control systems are known as being low-cost, flexible, scalable and adaptive. Additional to the requirements of a typical control system, a digital control system requires:

- An analog to digital converter (A/D) that has the function to convert the analog plant data into the discrete form in which the converted data will be delivered to the controller.
- A digital to analog converter (D/A) that has the function to convert the digital controller output data into the analog form in which the converted signal will be delivered to the plant.
- A software program that takes the input from A/D, computes the controller output, and sends the output to D/A.

Another aspect for digital processing, while using a computer with finite precision, we need to be careful to ensure the error in calculations.

A networked control system (NCS) is a system with real-time requirements in which the elements of the system are not physically located closely, but have the flexibility to be distributed as functional nodes to different locations. In a networked control system, a well-defined modular structure is needed to ensure scalability and performance of the system. Defining each node by a function allows functional scalability. The simplest way of describing such a modular system is Sensor-Controller-Actuator model made up of three computer nodes. Sensor node has the function to collect all the data from the plant that is necessary for the controller node computations. Controller node is responsible to compute the controller output signal based on the information received from the sensor and a control algorithm. Actuator node has the function to generate the necessary output signal to the plant based on the received controller output signal. A networked control system also requires a communication network to convey the information between the nodes.

Decentralized structure allows us to design various combinations of scalable nodes. As an example, in a case of multiple plants to be controlled by a cost effective

solution, decentralized structure allows us to design a network with multiple actuators and sensors and connect them to a single powerful controller, or use multiple sensor and actuator nodes to control a single plant.

In a networked control system, the output of the plant is periodically sampled by the actuator, encoded into a data packet and sent to the controller node. The controller node processes the received packet using a control algorithm and produces a control signal output, which is sent to the actuator node (Fig 1-1). The task of the actuator node is to apply the periodically received signals to the plant. However, since there are communication delays and data loss inherent on the communication network, not all packets make it to their destination on time, and some are lost on the way. Modern communication systems generally do not consider the delay as a problem, and remedy the loss situation by resending the lost data packet. However, in a networked control system, this means extra time is lost, and rather than resending the old data, it is better to transmit a more recent plant output sample, or a newer control signal instead. Similarly, delayed packets may be considered as lost since control signal applied late to the plant does not guarantee stability.

This thesis addresses a solution to this problem; the Model based Predictive Networked Control System (MBPNCS) method [1], [2] attempts to implement an experimental system to realize the method to control an actual plant.

In this thesis, we aimed to:

1. Make an implementation of the MBPNCS proposed by [1] which has been tested as a simulation before.
2. To achieve this aim, prepare a highly scalable and reliable NCS
3. To prepare a modular development environment that can be used in real NCS applications

The complete experimental apparatus used in this work consists of one sensor, one controller and one actuator node. Each implemented as a separate computer (of the type PC104) and the communication network is Ethernet (Figure 1-1).

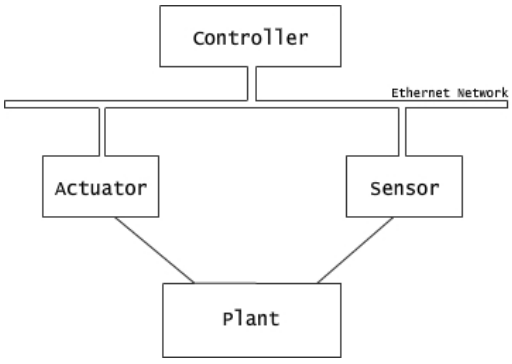

Figure 1-1 – Layout of the test bed

To provide for real-time performance of the computer nodes, a real-time operating system based on Real-time Linux was designed and implemented, and for the communication, real-time socket layer communication over Ethernet was implemented using UDP messaging. This does not ensure any real-time communication timing, but only is a means to remove the unnecessary delays within the computer node. As a plant, a motor of Maxon type 144501 is used. Speed control of the motor was accomplished by using a state feedback controller as the control algorithm on the controller node, and by measuring the complete state information of the motor consisting of its speed and current.

The thesis consists of six chapters. In chapter two, research on network problems of a networked control system is summarized. In chapter three, MBPNCS method used in this thesis is presented. In chapter four, implementation steps of a MBPNCS are explained. In chapter five, experimental work is described and test results are presented. And in chapter six, work done up to now is concluded and commented on the future work.

This thesis has been supported by TUBITAK in the scope of project 106E155.

2 BACKGROUND

2.1 Previous Work on Distributed Control Systems

Much work has been carried out recently with distributed control of large-scale systems. Computer networks and control theory have developed separately over the years and are reached at mature levels. Initial work on computer networks focused little on the topics of guaranteed delivery without retransmission or latency of the transmitted signal. Lost packets problem is solved with a confirmation response and retransmission of the signal when needed. For this reason, commonly used computer networks today are non-reliable with respect to these topics.

Control theory is developed in a centralized manner where communication between sub-structures is assumed to be free of signal loss or delay in transmission. Interaction between computer networks and control theory started with NCS and is an active research area. NCS is not a matured solution of the works in these areas. As a result, mixing those two theories through NCS introduced new problems.

In a NCS, delivery of signals from sensor node(s) to controller node(s) and from controller node(s) to actuator node(s) must be guaranteed for each sampling time to ensure the stability of the controlled system. For this reason, computer networks used commonly today for the practical applications are not suitable without presenting a robust solution to the NCS [3]. In this chapter, previously introduced methods to solve the communication problem in NCS are summarized and assumptions are explained with pros and cons.

2.2 Co-design of NCS

The aim of designing the network and control system together is to minimize the outputs of a given cost function. For this reason, network and control system should be designed together unless network cost function output alone is decreased to insignificantly small values using a given method. Previous works [4], [5], [6] defined cost functions for the packet loss and proposed methods to schedule NCS

transmission in the case of high loads. However, designing two systems together brings complexity and considerations for implementation. In the case of a change in one of those systems, redesigning might be needed when the change contradicts the considerations and assumptions.

2.3 Reduction of Communication

In the case that stability of the controlled system is affected by the rate of the loss of packets or delay in transmission decreasing the amount of network usage helps the linear time invariant approximation of the network. Proposed method [7] present methods to reduce the communication for various cases.

2.4 Deadbands

Deadbands offer a different type of reduction of communication to improve the network optimization [8]. If the packet to be sent contains identical or similar data with the previous packet, sender node does not transmit the packet. In the case of no packet transmission for a given period, receiver node uses the most recent one. The proposed solution assumes that the network is reliable and no packet loss is occurring.

2.5 Estimators

Estimators offer another method to reduce the communication load on the network [9]. Estimators use model of components on the receiver node for each sender node. The data to be received is estimated through models and communication network is only used for synchronization in normal operation. When the error between the estimated values and the actual values is above a threshold, the actual value is broadcasted over network and receiving nodes update their parameters when needed. This method depends on the reliability of the model however in the case of temporary loss of communication between the threshold value is exceeded and parameters are updated, stability of the controlled system is highly affected.

2.6 Disturbance Observer for the Network

To eliminate the effect of the time delay on the stability of the controlled system, the delay of the signal is considered as a disturbance [10]. This method is applicable to the system with fluctuant and unpredictable time delay and it has the same effectiveness as Smith predictor [11]. In [10], also controller design method considering disturbance was proposed.

2.7 Model Predictive Control (MPC)

In MPC, also a predictive model is used [12]. However, it is important to note that MPC assumes that a direct connection between the sensor node and controller node is established and the controller node has a-priori knowledge of the reference. To optimize the given controller variables, a cost function is defined. A model of the plant is then iterated with the control algorithm. Depending on the model output, the control output that will minimize the given cost function is chosen. MPC controller is combined of a networked control predictor and a conventional model predictive controller. The network control predictor, which selects the signal to be sent to the plant, is placed on the actuator node. The predictive controller uses the cost function as a reference to compute the control signals for the control horizon. Then, controller node sends all the computed signals to the actuator node.

In this chapter, a summarization of the previous works has been described and the next chapter explains the MBPNCS.

3 MODEL BASED PREDICTIVE NETWORKED CONTROL SYSTEM (MBPNCS)

The purpose of a MBPNCS is to bring a robust, stable and generic solution to the delay and loss of a packet problem of the distributed control systems. Delay or loss of a transmitted packet is not desired since it negatively affects the stability of the system. One approach to resolve this problem is to sustain the stability of the system by offering ways to minimize the delay in transmission. However, MBPNCS attacks the problem by compensating the missed data in the case of delay or loss of communication. In MBPNCS, neither messaging network nor the nodes are responsible to guarantee the timely delivery of the messages sent between nodes. When we mention about packet loss, both an actual packet loss and a delayed transmissions are referred hence the proposed method discards the delayed packets.

MBPNCS holds a model of the plant inside the controller and computes the next *n* predicted controller output states each period based on the plant model. The predicted controller outputs are appended to the controller output signal in a given period and sent to the actuator node.

The actuator node periodically applies the control output signal of the given period to the plant and keeps the predicted next state output signals in its buffer. In the case of a packet loss, which means a fresh data packet containing the control signal for the current period does not arrive; the actuator node uses the predicted plant output signal for the current instant that arrived with the last successful packet transmission. In the case of no packet loss, actuator node discards the predicted control signals that arrived with the previous packet in the next period. Below, MBPNCS nodes are explained in more detail.

3.1 Sensor Node

Function of the sensor node in MBPNCS is similar to the function of a sensor node in NCS. Assuming, time variable *k* is increased by one each period. At time *k*, sensor node periodically acquires the plant states *x(k)* and sends the acquired states to

the controller node. The sensor node does not necessarily receive any message from the other nodes. No mechanism is implemented to have an action in the case of packet loss.

3.2 Controller Node

The controller node holds the control function and model of the plant. Control function is responsible to compute the necessary control output signal that will be applied to the plant. Model of the plant is responsible to compute the estimated state of the plant for a defined number of future periods. For each iteration, estimated state of the plant is given to the control function and predicted control output signals are computed.

In a given period, the controller node is responsible to execute the control function and model of the plant and send the computed outputs of these functions to the actuator node. The source of input values to the control function and model of the plant in a given period depends on whether a packet is received from the sensor node or not. If the packet is received from the sensor node, the input source is the plant states received from the sensor node. If no packet is received from the sensor node, the input source is the predicted plant states computed by the model of the plant. The controller node is also responsible to notify the actuator node about the choice of input source.

3.2.1 Control Function

At time *k*, the controller node computes the control output *u(k)* using a suitable control method. In this work, feedback control was used.

$$u(k) = Kx(k) \quad (1)$$

u(k) is added to the message which will be sent to the actuator node.

3.2.2 Model of the Plant

The model of the plant resides within the controller node. Here, linear model equations were used, presented in state space form:

$$x(k+1) = Ax(k) + Bu(k)$$
$$y(k+1) = Cx(k) \quad (2)$$

On the controller node, when a plant state is obtained from the sensor node, which is expected for each period, predicted controller output signals are computed using the plant model for the next *n* state using the equation:

$$\hat{x}(k+n) = A'\hat{x}(k+n-1) + B'u(k+n-1)$$
$$\hat{y}(k+n-1) = C'\hat{x}(k+n-1) \quad (3)$$

At time *k*, control signal *u(k)* applied to the plant is applied to the model. The output of the model $\tilde{x}(k+1)$ is the estimation of the plant at time *k*+1. To compute the controller output at time *k*+1, control algorithm is applied to the estimated states $\tilde{x}(k+1)$. The control output $\tilde{u}(k+1)$ is then applied to the model of the plant to compute the next predicted output of the plant. This process is recursively applied *n* times and computed control outputs from $\tilde{u}(k+1)$ to $\tilde{u}(k+n)$ are sent to the actuator node including *u(k)*. The error between the model estimations and the real plant output can be defined as

$$\tilde{x}(k) = x(k) - \hat{x}(k)$$
$$\tilde{x}(k+n) = [(A+BK)^n - (A'-B'K)^n]x(k) \quad (4)$$

The proposed state space *A, B, C* matrixes and control value *K* must guarantee that $\tilde{x}(k+n)$ has an upper bound. Ideally $\tilde{x}(k+n)$ should be close to zero.

3.2.3 Sensor Flag

The controller node holds a sensor flag parameter, which will actually be used by the actuator node, included with each packet directed to the actuator node. At time *k*, if no packet loss occurs between the controller node and the sensor node, the controller node sets the sensor flag parameter up and works as defined above by computing the control output signal *u(k)* and predicted control output signals from $\tilde{u}(k+1)$ to $\tilde{u}(k+n)$ using the plant states *x(k)* received from the sensor node.

At time *k*, in the case of a packet loss between the controller node and the sensor node, the controller node computes the *u(k)* and $\tilde{u}(k+1)$ to $\tilde{u}(k+n)$ using the predicted plant state $\hat{x}(k)$ computed at time *k-1* and sets the sensor flag parameter down. Since predicted plant states are used to compute *u(k)*, control signal output is

less reliable in comparison to the control signal output computed with the real plant states. If the packet loss events consecutively follow each other, reliability of the computed control signal output and predicted control signal outputs decrease each period with a rate of $\tilde{x}(k)$. To overcome this reliability problem, sensor flag parameter is introduced sent by the controller node and a state transition diagram is prepared to run on the actuator node.

3.3 Actuator Node

In the case of no packet loss, the actuator node applies the control output signal *u(k)* received from the controller node to the plant. In the case of a packet loss, the actuator node uses a deterministic state transition diagram, Figure 3-1, to decide which control output signal should be applied to the plant.

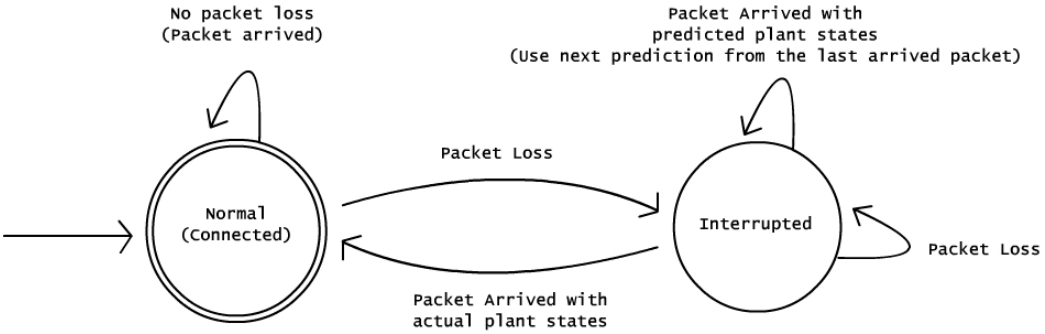

Figure 3-1 – State Diagram of the Actuator Node

In the state transition diagram, there are two states defined, namely “Normal” and “Interrupted”. Start state of the state transition diagram is the “Normal” state. Each transition is triggered with a packet loss or no packet loss event. When the actuator is in the “Normal” state, if a packet loss occurs, the actuator node goes into an interrupted mode, starts to apply the predicted control output signal $\tilde{u}(k+1)$ to the plant, which is received with the last successful packet transmission from the controller node.

The model of the plant located on the controller node, computes predicted control output signals for the next *n* state each period. For the following periods, the actuator node continues to apply the last predicted control output signal on top of the

stack, $\tilde{u}(k+n)$, of the same packet till it receives a packet from the controller node carrying a high sensor flag. While in the "Interrupted" state, if the actuator node receives a packet with predicted plant states flag instead of actual plant states flag, it discards the packet.

4 IMPLEMENTATION OF A MBPNCS

The main aim of this work was to implement and test the MBPNCS. In order to achieve this, the software to run the MBPNCS, the necessary hardware setup for the controller environment and a communication network needed to be prepared. There were many commercial solutions that provide an end-to-end networked control system that would save quite a considerable time in this work. However, it was preferred to prepare our own equipment and software as much as possible to understand the dynamics and to know the inner workings of the system in order to have a better understanding for the causes of the packet loss and performance of our method.

The work consists of several components. First a real-time operating system based on the real-time Linux kernel was implemented. Then real-time socket layer communication method was implemented. Another step was to design and build the physical controller setup and electronic hardware for interfacing the various parts and measurements. The device drivers for the electronics components were written, and the physical parameters of the plant were measured using several techniques, and the model constructed. Finally the MBPNCS system software was ported to the system described. Since the simulations of MBPNCS were done at a low level of abstraction, most of the code from the simulation could directly be used in the implementation. Each step is explained in detail below.

4.1 Design and Implementation of a Real-Time Embedded Operating System

When we talk about high-level software development for embedded systems, operating systems come in to the picture. Assurance to meet the requirements and functional expectations for an embedded operating system are mostly application specific. On the other hand, the most common requirements for such an industrial system are compactness, ability to serve for long-term period without any manual intervention, providing real-time capabilities and ability to recover from failure.

Though many commercial solutions exist to be used in this work, we preferred to prepare our own operating system from scratch using the Real-Time Linux kernel, RT Linux. RT Linux has some well-known favorable features as being open-source, portable and reliable. The monolithic kernel is easily reconfigurable with widely available functionalities. The final version of the prepared real-time Linux distribution is named TuxSA.

Since, the common configuration of Linux kernel is prepared to work on a PC, we were forced to understand the inner workings of the Linux and reconfigure it as an embedded system. During reconfiguration, we experienced various technical problems however the Linux loader program, LILO, which is executed initially after boot-up, was the main source of time-consuming problems. Since the image of the operating system is prepared in a different development server, the LILO needed to be configured carefully with respect to the geometry of the disk, primary disk configuration differences between the server and the PC104 and kernel image pointers. To make the operating system work in the PC104 platform other than the development server, cross configurations were needed to be studied deeply to solve the technical conflicts.

4.1.1 Operation from RAM disk

The weakest point of Linux that results in failure is the way it manages a file system. In the case of a power failure or a kernel panic, it is highly probable for a non-recoverable corruption occurs in the file system. Using a journalled file system also does not always guarantee the recovery from a failure and has additional costs to operate. On the other hand, the PC104 type computers used in this thesis provide compact flash (CF) disk as the default storage and CF disks are known to have limited erase/write cycles. Simply installing Linux on CF disk means that after a certain run time, the disk hardware would fail (in our implementation, the failure took about three days of continuous operation). To assure the ability to recover from failure and overcome the limitation of CF disks, RAM disk method is used.

Kernel program, the files necessary to run the operating system, helper programs and controller system modules are placed on the CF disk in a compressed compact form. After the kernel boots up, a file system (RAM disk) of 64 MB is created on the system RAM. Then, the files from the CF disk are extracted into the

ram disk. The next coming booting procedures continue on the ram disk. After the booting is completed, installed controller system module starts to run. From booting up to run-time, file system on the CF disk remains as read-only that ensures that no corruption happens in the core.

4.1.2 Real-Time Performance Guarantee: RTLinux

A digital control system works in periodic cycles and period of the cycle is a performance affecting issue from the control point of view. Previous tests performed in Matlab have shown that the discrete control system with 10^{-3} seconds of sampling time works under stable conditions with the prepared controller and plant.

There is no native framework that supports periodic real-time task creation in Linux kernel. For this purpose, open source version of the RT-Linux 3.2pre3 (This version is not available on the official Internet site anymore) is used. When RT-Linux is enabled on the runtime, it works as a microkernel and starts to manage the CPU. Linux kernel is behaved as the lowest priority task and executed when no real-time task is pending in the queue. Since native Linux sockets are used for communication, guaranteeing execution of the Linux kernel for the sake of the packet transmission is also a consideration.

The PC104 computers used in this work have a 300 MHz AMD Geode processor installed. Jitter analysis test has been performed on these PC104's by writing a periodic program generating square signals at the parallel port with a period of 10^{-3} seconds. Data for a total of 24 hours is gathered via oscilloscope and jitter is calculated. Calculated worst-case jitter was 183 μs. The same test was performed on a PC with Intel Celeron CPU and calculated worst-case jitter was 24 μs. Also, it has been reported by the FSM Labs that the best performance of a PC104 with RTLinuxPro 1.2 is 104 μs [13]. Based on these tests, we concluded that, for the sake of system performance, an embedded system with a better performing CPU and architecture would better be chosen. However, we continued to work with the available PC104's.

On the other hand, it is important to note that tests shown that the 300Mhz AMD Geode processor and PC104 architecture can go up to a sampling time of 10^{-4} seconds with the same jitter value while the processing load of the sensor and actuator

is the same and the controller's processing load is only PID computation using floating numbers (no predictive model computation). Therefore architecture choice will become a design issue when we are using a system with a sampling time of 10^{-4} seconds and the system also runs heavy computations like modeling or kinematics equations.

4.1.3 Communication

4.1.3.1 Medium

Ethernet is chosen as the communication network since it is widely used in many applications and supported as default by PC104. A non-reliable network environment is needed since data loss and delay scenarios are needed for the tests. Using an Ethernet switch as the common connection point does not provide this requirement since each connected device has its own collision domain. For this reason, a typical hub is used as the common connection point.

4.1.3.2 Messaging Protocol

Messages between control system nodes are delivered by IP packets. For this purpose, an IP network is established between PC104's (Figure 4-1) and packet transfers are handled using the existing socket functions in the Linux kernel.

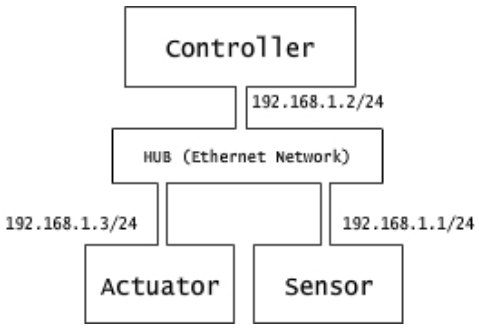

Figure 4-1 – Established IP Network

In a distributed control system, the accuracy of the timing of the communication between control nodes is vital. Therefore, in the case of a delay in the

communication, the packet is accepted as lost and discarded even received. For this reason, UDP is used as the message passing protocol. Using a guaranteed real-time protocol may be an idea however such a protocol needs a supported network environment with strict constraints, is only commercially available and does not solve the temporary loss of communication problem. On the other hand, the aim of this thesis is to confirm that there is a robust solution to the temporary loss of communication problem.

RTLinux allows non-real-time packet transmission over network in a real-time application via the RTSock framework. RTSock framework uses the standard TCP framework of the Linux operating system. In order for RTSock to function, it needs a program running in the user space to create a UDP socket. User space program binds the created UDP socket to a special FIFO (/dev/rtsock) so that the RTSock framework running in the kernel space can directly manage it. This method has the advantage to use all the facilities Linux kernel provides natively however; we must ensure that the CPU utilization is sufficient enough for the low level Linux kernel to be executed.

RTSock is not yet a complete library and it does not provide a robust environment. This drawback makes RTSock hard to use in a practical application. During tests, we experienced many technical difficulties with unresolved root causes.

We experienced an important lack of robustness in RTSock during the final tests. We discovered that Sensor node stops sending packets at random times in each test and RTSock is giving an error because of losing the socket buffer. In normal conditions, losing a socket buffer on the memory is not possible and unexpected. The issue turned out to be an interoperability problem between RTLinux and RTSock and is related with the usleep function used in encoder readings and ADC operations. Usleep is a native Linux and RTLinux function that yields the process with a given amount of time in terms of μs. However the Linux and RTLinux versions of the same function are coded in a different way.

Native Linux usleep function yields the process and wakes up the process using a timer. Minimum amount of time for a successful yield operation for a standard Linux user space process is 2 ms. Minimum amount of time for a successful yield operation with usleep for a standard Linux user space process is 10 ms. The 10 ms lower bound is a design issue and it is set to guarantee to do precise timing in FIFO tasks. In the case that less than 10 ms is given as a parameter value to the usleep function, instead of yielding the processor, the function goes into a busy-wait state.

On the other hand, native RTLinux usleep function is different from the native Linux usleep function.

RTLinux has a function named pthread_wait_np that yields the processor for a real-time process and wakes the real-time task up in the next period. Pthread_wait_np is an important function from the point of real-time constraints that the timing of wake-up can be directly associated with the real-time performance. Minimum time for a successful yield operation for pthread_wait_np is architecture dependant and is 30 μs with the PC104 in use. Using a period time below 30 μs results in unpredictable behaviors in the system.

Usleep function in RTLinux is almost the same with pthread_wait_np except the wake-up timer is set to the given value. Therefore, minimum time for a successful yield operation for usleep is also architecture dependant and is bounded by the minimum successful yielding time of the pthread_wait_np function, which is 30 μs in this case. However, it has been noticed that usleep is used for 10 μs in encoder reading and ADC operations. Setting usleep to 10 μs is acceptable from the view of hardware operations and yielding the processor more is not necessary. However, using 10 μs for usleep resulted in unpredictable behaviors in the system and somehow made the RTSock to lose the socket buffers in the memory.

Providing a real solution to this issue requires altering the native RTLinux and RTSock code and far beyond the scope of this thesis. Setting the usleep time to 30 μs for encoder reading and ADC operations solved the issue as a workaround.

4.1.3.3 Utilization of the Network

The maximum utilization of the Ethernet network by PC104 nodes is calculated using an open source bandwidth measurement tool, Iperf [14]. The test application is run on two PC104's in a client-server style for 100 seconds. No disk reading operation is done during the test to avoid the upper bound speed rate of a disk reading operation. The data send was random 1's and 0's generated by the CPU. The maximum utilization observed from this test was 74 %.

The number of predictions we can generate in a period is bound by the computation power available and carrier capacity of the network. Each computed prediction state consumes 4 bytes both in memory and in an IP packet (Assuming the size of a float type variable in C is 4 bytes). Ethernet has 14 bytes of header per data

package that may be omitted. Assuming we have no computation power limitation using a 100Mbit Ethernet (bandwidth, bw, in bytes/s) network with 70 % utilization (ut) and a sampling time of 10^{-3} (ts) on the control side and br denoting bytes occupied in the packet by a single prediction, equation 5 calculates maximum number of predictions (mnp) that can be carried on the network.

$$mnp = bw*ut*ts / br \quad (5)$$

With the given bandwidth, utilization and sampling time values above, the network has the capacity to carry 2187 predictions per period. That means the system is fault tolerant to communication loss up to 2.187 seconds in our implementation. When the communication is lost for more than 2.187 seconds and the actuator is out of predictions, the last prediction of the finally received packet will continue to be applied to the plant.

Another possible issue is fragmentation of the IP packages. Assuming that 2187 predictions are computed per period, which consumes 8748 bytes, the data will be sent in at least 6 packets per period because of the limitation of maximum 1500 bytes per IP packet [15]. Loss of any of the 6 packets means loss of all, which decreases the stability of the system. This consideration will become an issue when the system computes more than 375 predictions (1500 bytes) per period. Using TCP may be a solution if the receiving time of retransmitted lost packet is guaranteed in the given period however that assurance is hard to achieve in practice. In the case of UDP, 6 times larger IPv6 jumbograms should be used whenever possible. However, for prediction data that is over 9000 bytes per period, fragmentation is obligatory since CRC field in Ethernet is effective only up to 9000. For more information on this issue, please check RFC-2147 document [16].

The CPU power necessary to compute the defined predictions is related to many parameters like defined CPU instructions, resource requirements of the other programs running on the machine, programming language choice, the way algorithm is written and compiler optimizations. Tests with the current PC104 shown that at most 200 predictions can be computed in one millisecond on Geode AMD 300Mhz processor with the current code sequence. However, the code is not optimized fully for the processor and a generic GNU compiler is used.

All of the mentioned above affect the system performance of the platform designed in this thesis and should be taken into consideration for practical applications.

4.1.4 Runtime Data Storage and Retrieval of Experiment Results

Any data saved to the disk will be lost in case of a power failure since TuxSA Linux runs on a ram disk and compact flash disk is dismounted in normal conditions. Data gathered from kernel modules can be passed to user space programs using FIFO's. Any data gathered while the system is running can be stored permanently in two ways.

1. Transmission over network

   Sampled data can be transmitted to other network nodes, which are capable to permanently save the data.

2. Using the CF disk

   Though ram disk is used by the operating system, it is possible to reach the CF disk in runtime. This method has two drawbacks. First, CF's have limited storage capacity (Currently, maximum 8 Gigabytes). CF's used in this work had 128 Megabytes of storage capacity and 9.2 Megabytes of the CF are already occupied by TuxSA. Second, if a failure happens while the CF disk is mounted in writable mode, stability of the file system located on the CF cannot be guaranteed. In the case of a need to a longer time data saving than the CF can handle or stability is a high priority, the method of transmission to another node on the network should be used.

4.1.5 Runtime System Monitoring and Intervention

The prepared system supplies local and remote monitoring. Intervention to the running-system is possible using FIFO communication. Local monitoring is possible by connecting a monitor and keyboard to the PC104's. It is also possible to remote

monitor the system and to give new commands in run-time over a display and input unit connected to the same network.

4.2 Device Drivers

To implement the system, several interface modules had to be used such as analog to digital controllers, digital to analog controllers and quadrature encoder readers. The driver software for each of these had to be written, since they are not available under the Linux operating system.

4.2.1 A/D Conversion

Analog to digital conversion is needed for the sensor node to take the analog input from the plant and send the necessary computed information to the controller. Requirement for such an A/D board is to have 12-bit resolution on the input. The A/D function of KONTRON ADIO128 module is used for this purpose. KONTRON ADIO128 has no Linux driver published by the manufacturer. Therefore, kernel driver of the A/D function is prepared for the device. A/D conversion value of a channel is written to two bytes in a high byte-low byte format. The hardware can measure a maximum of ±10 Volts. Following table is the observed values for +10 Volts, 0 Volt and –10 Volts.

Input Voltage	High Byte (Hex)	Low Byte (Hex)
+10 Volts	0x10	0x0F
0 Volt	0x00	0x00
-10 Volts	0xF4	0x01

It is also noted from the user manual of the device that a noisy signal requires more integration and measurements have a 0.2 % upper error bound [17].

4.2.2 Quadrature Decoder/Counter

A quadrature decoder is needed for the sensor node to resolve the quadrature encoder shaft position inputs that are generated by a circuit bundled in the Maxon motor to monitor the motor shaft rotations. A fifteen channel MSI P400 module board is used for this purpose. MSI P400 has no Linux driver published by the manufacturer. Therefore, kernel driver is prepared for the device. MSI P400 associates a counter within each channel that has a range of 0-65535 and has a high byte-low byte representation on the I/O access. This counter is updated continuously after power on and no triggering function is required to read the counters. Therefore, it is possible to read the counters while its being updated that might result in incorrect data gathering. To prevent this condition, in a single counter read operation, two consecutive read operations is done on the high byte and compared to each other. If both read operations result the same, the data gathered is accepted as valid. Consecutive reads are repeated till the sampled data is marked as valid.

After consecutive read solution is implemented, the card is tested and it has been observed that in a given period, 81.8 % of the data marked as valid have an error with an upper bound of 2 %. Also, 18.2 % of the data marked as valid turns out to have an error with a lower error bound of 2 % and upper error bound of 20 % when the plant is in the steady state. Since deviations up to 20 % is not desired in such a system, we tried to find the root cause of this condition though the supplier of the product did not give much information about the inner workings of the hardware. The root cause of this issue has not yet been found but solved with a workaround. Instead of doing a single read operation in a given period, multiple read operations and total summing is proposed. 10 multiple read operations are executed with an interval of 30 μs in a given period (1 milliseconds). After this fix is applied to the code, the rate of valid data that has an error between 2 % and 20 % is decreased from 18.2 % to 3.7 %. As a final result, 96.3 % of the sampled data has an error with an upper bound of 2 %.

4.2.3 D/A Conversion

Digital to analog conversion is needed for the actuator node to send the digital output received from the controller node to the plant. Requirement for such a D/A device is to be able to create voltage between –10 and +10 volts. The D/A function of

KONTRON ADIO128 module is used for this purpose, which has four output channels. One output channel is used for the plant. KONTRON ADIO128 has no Linux driver published by the manufacturer. Therefore, kernel driver of the D/A function is prepared for the device.

4.3 Hardware Preparations and Setup

For the plant, no considerations were defined and a typical DC motor (Maxon 144501) is used. Additional circuits were needed to be prepared during the platform setup. A current reader circuit is prepared for the sensor node to read the current of the plant. Also a DC motor linear power amplifier circuit is prepared for the control output signal. In Figure 4-2, we can see the real platform setup visually. One of the PC104 computers can be seen on the left upper part. Power supply is on the right upper part. Motors are located in the middle and intermediate circuits are placed to proper locations on the platform.

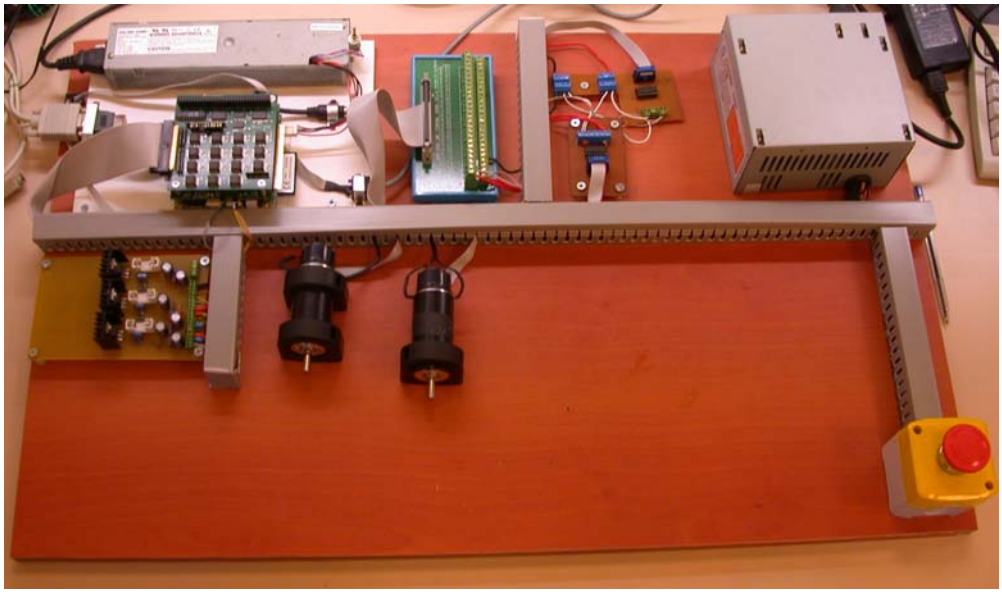

Figure 4-2 - Platform Setup

4.3.1 Analog Input and Output: Kontron ADIO128

We used two Kontron ADIO128's that have the function of an ADC on sensor node and a DAC on the actuator node. The ground connections of both devices are connected to each other to have the same ground base. Kontron ADIO128 has a cumulative drive current limit of ±20mA due to the limitation of PC104 power supply. Because of this limitation, only two DAC channels can be used at the same time since a single DAC channel can drive up to ±10mA [17]. To overcome the limitation, the board supports additional pins to supply external ±12 Volts.

Channel 1 of ADC on sensor node is connected to a circuit operation amplifier based current sensing circuit. Control signals are carried through channel 0 of DAC on actuator node, which is connected to the driver of the motor. The amplifiers in the driver circuit are class A/B (linear), which results in small differences in the output with respect to the reference output because of internal offset. The offset of the driver circuit might be measured and added to the calculations of the DAC function.

4.3.2 Quadrature Decoder: MSI P400

Sensor node is equipped with a quadrature decoder to acquire the position information from the plant. One input channel is used for the plant and other available (fourteen) channels are reserved for future work when more plants are added to the system.

4.4 Sensor, Controller and Actuator Nodes MBPNCS Software

Each node of the distributed control system is placed on a separate PC104. A typical PC uses PCI computer bus and ATX form factor. However PC104 has its own form factor and computer bus definition that is designed for industrial applications. A PC104 has no backplane and designed as stacking the necessary modules on top of each other. In this work, each PC104 is equipped with the necessary add-on modules.

Nodes of the control system are implemented as programs running in the kernel space. All the software prepared in this work is written in C language because of kernel space requirement. Each node is automatically loaded to the kernel after

booting. However controller and actuator nodes wait in standby mode till they receive their first message.

Synchronization of the timers in each node is needed since nodes run as periodic tasks and generated message in a period is only valid for that given period. Since message flow starts from the sensor node and continues with the controller node, clock of the sensor node is chosen as the master clock. Master clock indirectly depends on the system clock. Execution steps of the synchronization algorithm and packet discard policy is described below:

Initialization:

1. Master clock of the sensor node is set to 0 milliseconds
2. Time counters of the controller and actuator node waits in standby
3. Sensor node sends the clock information with the first message to the controller node
4. Controller node updates the time counter and becomes active
5. Controller node sends the updated clock information with the first message to the actuator node
6. Actuator node updates the time counter and becomes active

For each period

7. Time counters are incremented in each node
8. Sensor node appends the clock information to the sensor data message directed to the controller node
9. Controller node checks if the received master clock is same with the local clock
10. If not equal, the packet is discarded and the local clock is updated
11. If equal, packet is accepted.
12. Controller node appends the clock information to the controller output data message directed to the actuator node
13. Actuator node checks if the received master clock is same with the local clock
14. If not equal, the packet is discarded and the local clock is updated

15. If equal, packet is accepted.

During the tests, an issue has been observed that the clocks between the nodes become out of synchronization in each period and needed to be synchronized respectively. Since clock updates and packet sending is done in different threads, difference in execution time and jitter of the real-time periods result in difference in the clocks. Difference in execution time is altered by putting the clock updating code just after the start of the next period. Since periods are fixed, the difference in timing to update the clocks now has an upper bound same as the jitter. Guaranteeing such an upper bound is sufficient for synchronized execution of local clocks between nodes.

Current synchronization method has an important handicap. In the case of a mismatch between the local clock and the received reference clock, since the reference clock is gathered with the received package, we cannot differentiate whether the reason of mismatch is because of the delayed packet or synchronization error. To overcome this, clock beats should be received over a different channel like an ADC line connected to all nodes or using GPS clock synchronization.

5 EXPERIMENTAL WORK

Before applying the tests, a parametric system model is prepared using the measurements made on the plant for characterization. Later, model parameters are optimized using hill-climbing method to optimize the model. After preparing the model, 6 tests are applied on the system with various packet loss rates.

5.1 Derivation of the Plant Model

Method decision for system identification in model predictive control needs to be studied beyond the scope of this thesis. Linear model predictive identification methods are introduced since the beginning of 70's and well studied. Non-linear model predictive identification is still an active research area. Modeling method decision is mostly area specific that depends on various considerations. As an example, the method choice depends on the representativeness of the sample data with respect to the real plant. Previous works approached this problem by defining a model confidence index.

A MBPNCS requires modeling of the plant to be used in the controller. For the Maxon motor type 144501 used in this work, motor parameters below which are published by the manufacturer are used as a reference. PID, control parameters are tuned in Matlab for stability of the system using root locus. When the prepared model and control parameters are applied to the real system, unstable behaviors and high error rates are observed. Since parameters given by the manufacturer turned out to be not sufficiently accurate, system identification methods are used to prepare a stable model. For this purpose, step input is given to the motor and response of the motor is recorded. Sampling is done 20 times to have a more representative data space. This data is processed under system identification toolbox of Matlab using parametric models in state space structure.

When we consider the "Model Variables - Error of the Model" space for the obtained model, we assume that the variables of the model generated by the parametric methods obtain a location close to the global minima. In Matlab, PID

control parameters are optimized using root locus. Later, hill-climbing technique is used for offline optimization. Parameter space is quite big to search for a combination of A and B matrixes. Hill-climbing 100 times for each variable means visiting 10^{10} points in the variable space. To use the available processing power efficient, 10 iterations are done for each variable with different ranges using the original values as a reference point.

State space representation is conserved for the model during training. Produced model has a root mean square error (RMS) of 1.405 (Predicted Speed – Actual Speed) on runtime with step input. Figure 5-1 shows the step response of the model and compares it with the real plant.

R = 11.80[Ohm]	Terminal resistance
L = 3.1631*10-3[Henry]	Terminal inductance
Km = 128.03 * 10-3[Nm/A]	Torque constant
Kv = 773.5[rad/sV]	Speed constant
J = 4*0.01[kgm2]	Rotor inertia
b = 1.571*10-8[Nms/rad]	Friction coefficient

Parameters above given by the manufacturer, have been used in the speed model below.

$$A = \begin{bmatrix} -R/L & -Kv/L \\ Km/J & -B/J \end{bmatrix} \quad (6)$$

$$B = \begin{bmatrix} 1/L \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

$$D = 0$$

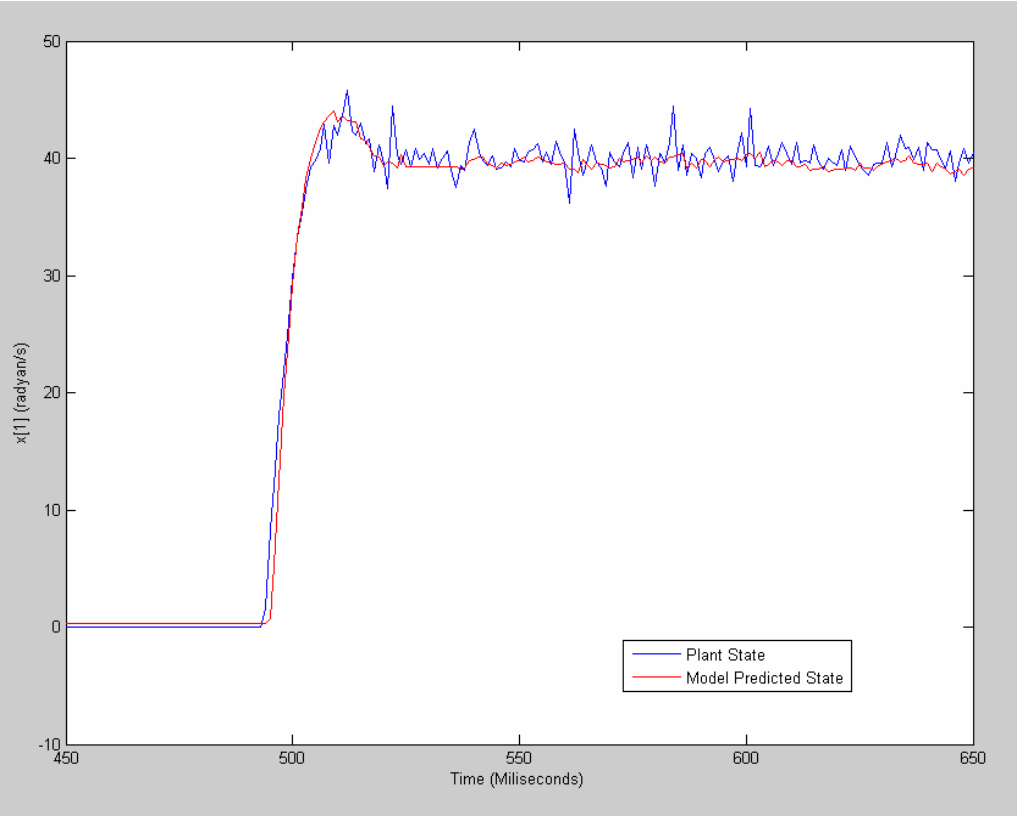

Figure 5-1 – Plant States and Model Predicted States for Step Response

To cancel the negative effects of the distributed system on the test results, verification of the model is done in a centralized control environment using a single node of the distributed system. Hardware modules with necessary (DAC, ADC) functions are installed on the central node for this test. In the verification setup, model and plant are fed with the same controller output values while the deviation of the model from the real plant is calculated with a mean square error function (Figure 5-2). The computation of $\hat{x}(k+1)$ in the model is not applied to the control equation, but rather it is used to calculate RMS error of the model.

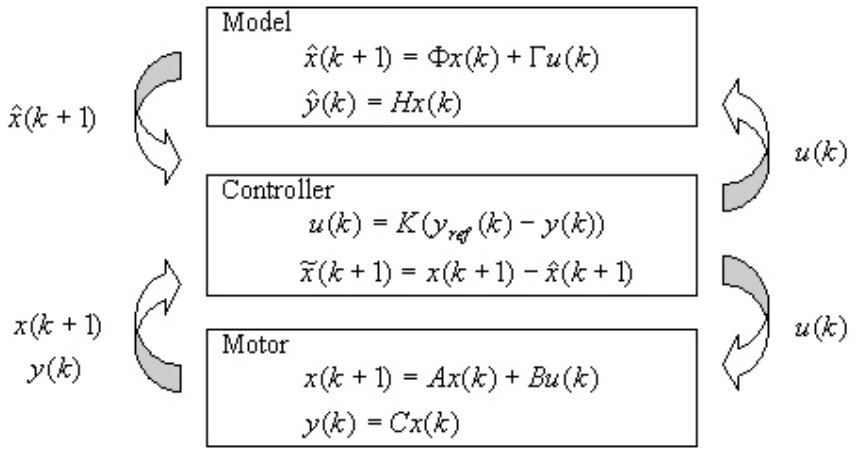

Figure 5-2 – Online Model Verification Setup

After discretizing the continuous time system with a sampling time of 0.001 seconds and online verifying the discretized model, we used the calculated model parameters shown below for the tests.

$$A = \begin{bmatrix} -0.02319 & -46.3821 \\ 0.00084 & 0.75630 \end{bmatrix}$$

$$B = \begin{bmatrix} -463.821 \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

$$D = 0$$

5.2 Performance of the implementation of MBPNCS

Tests are divided into two categories by the type of behavior of the packet loss. First category presents the results with random packet losses following a pattern to represent a network transmission with non-stochastic failures. By pattern following, we mean that the order of the number of consecutive packet losses repeats itself periodically (every ~50-250 milliseconds). Second category presents the results with random packet losses not following a pattern to represent a network transmission with

stochastic failures. The aim to define such categories is to observe the control system stability under different class of network failures.

Category 1 is divided into five cases in terms of loss of packets. First case is applied without any manual intervention that describes the natural behavior of the current system. Former cases are based on randomly created packet losses with a packet loss ratio of 50 %, 70 %, 90 % and 98 % in order. Packet loss randomization is needed for the tests but RT Linux library does not support randomization functions. Therefore a simple almost randomizing algorithm is used. Packet loss randomization algorithm works on a basic pseudorandom number generator where the multiplier is calculated and taking the mod of the multiplied number. If the resulting number is above a defined threshold, the packet is dropped, if the resulting number is below the threshold, the packet is accepted. Below is the pseudo code of the packet loss decision algorithm. Given values of Multiplier, Mod and threshold are for 98 % packet loss as an example. Giving different values to Multiplier, Mod and threshold result in different rate of packet loss.

```
Multiplier = 11
Mod = 25
Threshold = 0
For each periodNumber from 0 to ~ (As long as the system is running)
        Multiplier = ( Multiplier * Multiplier ) + periodNumber
        tmp = floor( Multiplier / Mod )
        Multiplier = Multiplier – ( tmp * Mod )
        If( Multiplier > Threshold )
                Drop The Packet
        End
        Else
                Accept The Packet
        End
End
```

For all of the cases, 65 predictions are generated per period. Plots show the reference speed and actual plant speed. The number of consecutive packet loss is added to the plot with –40 to -100 offset for better visibility.

5.2.1 Category One: Non-stochastic Packet Losses

Case 1.1: Natural Behavior

In this test the aim was to observe the performance of the MBPNCS without any manual intervention in terms of packet loss. In other words, on the network, only the control signals existed. In this test, 0.33 % packet loss is observed on the network.

Red line plots the reference speed given to the controller. Blue line plots the measured actual speed of the plant and black line plots the number of consecutive packet losses on the network. For example, if the value of the black line is 0 (Plotted as –40), that means the packet for the given period is not lost. If the value of the black line is 8 (Plotted as –32), that means last 8 packets are lost.

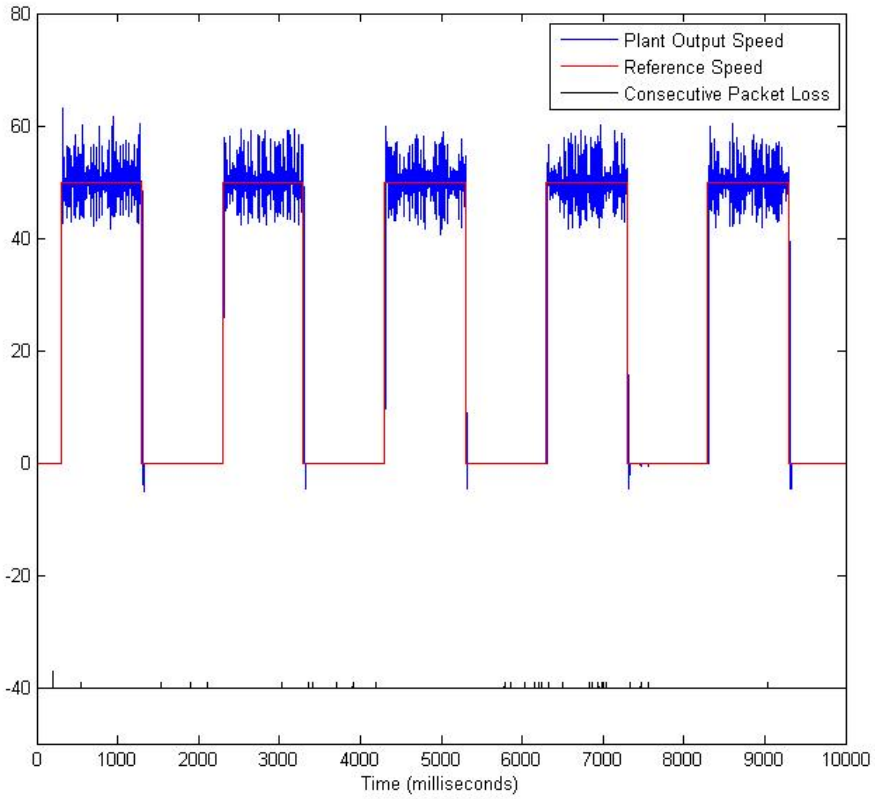

Figure 5-3 – 0.33 % Packet Loss - MBPNCS

RMS error of case 1.1 is 1.6247. Overshoot is 16 % and undershoot is 0 % for step response.

Figure 5-4 is the detailed view of a region from Figure 5-3 where the reference speed is changed from 0 radian/s to 50 radian/s.

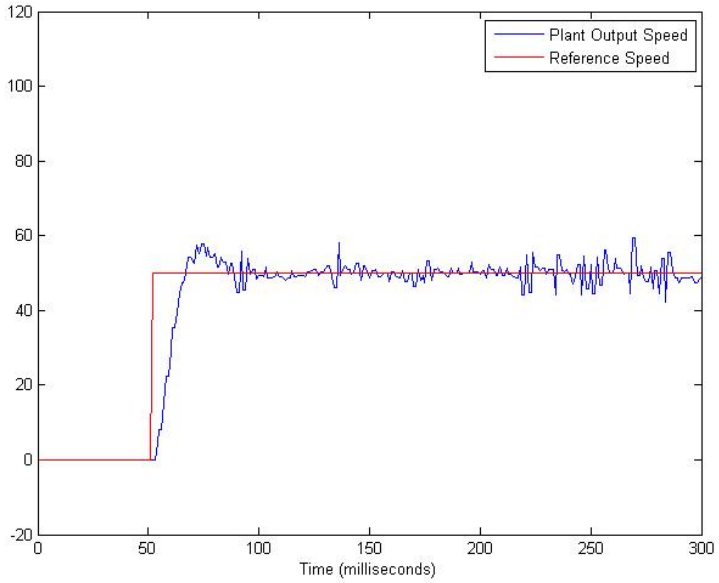

Figure 5-4 – Detailed View of a Step-Up - 0.33 % Packet Loss – MBPNCS

Plant output was shifted four periods back in detailed views since plant output is sampled in the controller node. Plant output sampled in the controller node belongs to the reference of the past fourth period. Figure 5-5 is the detailed view of a region from Figure 5-3 where the reference speed is changed from 50 radian/s to 0 radian/s.

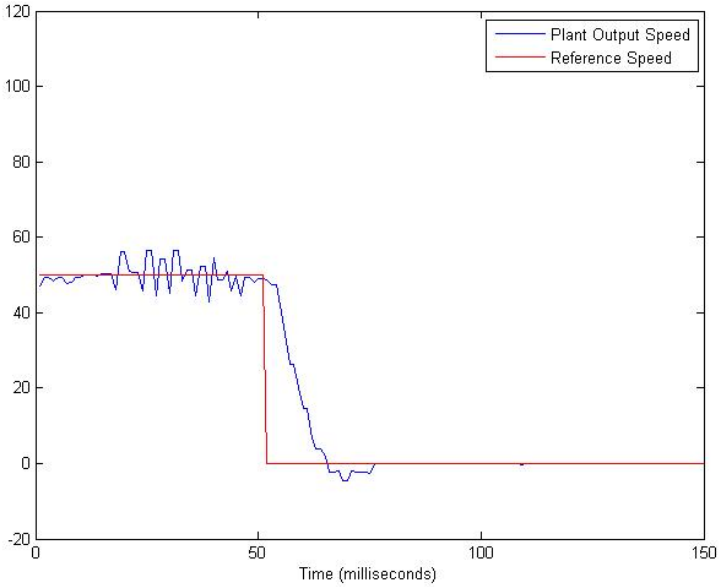

Figure 5-5 – Detailed View of a Step-Down - 0.33 % Packet Loss – MBPNCS

Case 1.2: 50 % packet loss.

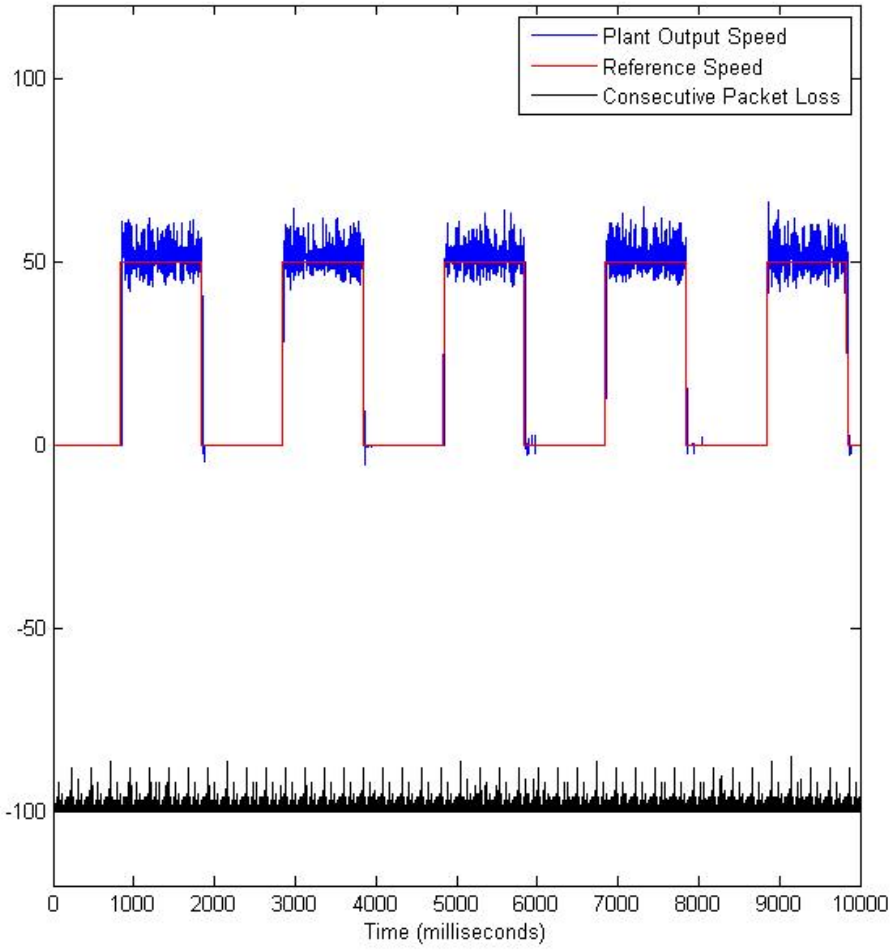

Figure 5-6 – 50 % Packet Loss - MBPNCS

RMS error of case 1.2 is 2.1035.

The consecutive packet loss plotting can be interpreted as simulating a congested network. The oscillation of the plant speed is due to measurement noise of the encoder reading operation since we did not use any kind of filters. The 3.7 % error rate in encoder reading explained in section 4.2.2 also has an effect on the oscillation. Using a low-pass filter as a future work will decrease the oscillation. It is also important to note that oscillation is not observed while measuring the actual speed using a tachometer. Tachometer displays mean of multiple reading operations in a fixed period.

Figure 5-7 is the detailed view of a region from Figure 5-6 where the reference speed is changed from 0 radian/s to 50 radian/s.

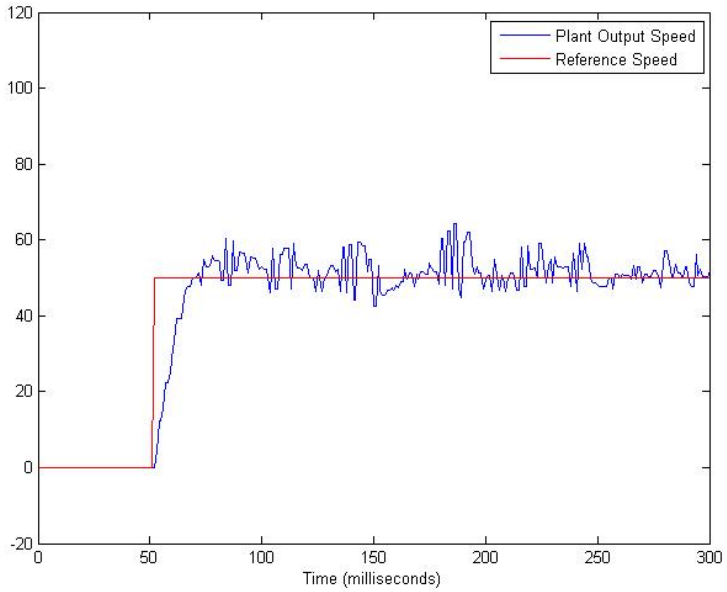

Figure 5-7 – Detailed View of a Step-Up - 50 % Packet Loss – MBPNCS

Overshoot can be decreased with tuning the proportional gain parameter. Figure 5-8 is the detailed view of a region from Figure 5-6 where the reference speed is changed from 50 radian/s to 0 radian/s.

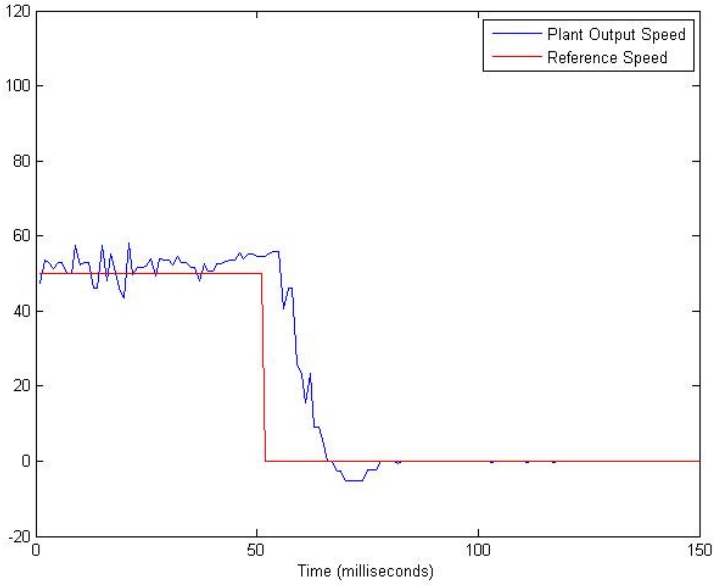

Figure 5-8 – Detailed View of a Step-Down- 50 % Packet Loss – MBPNCS

Case 1.3: 70 % packet loss

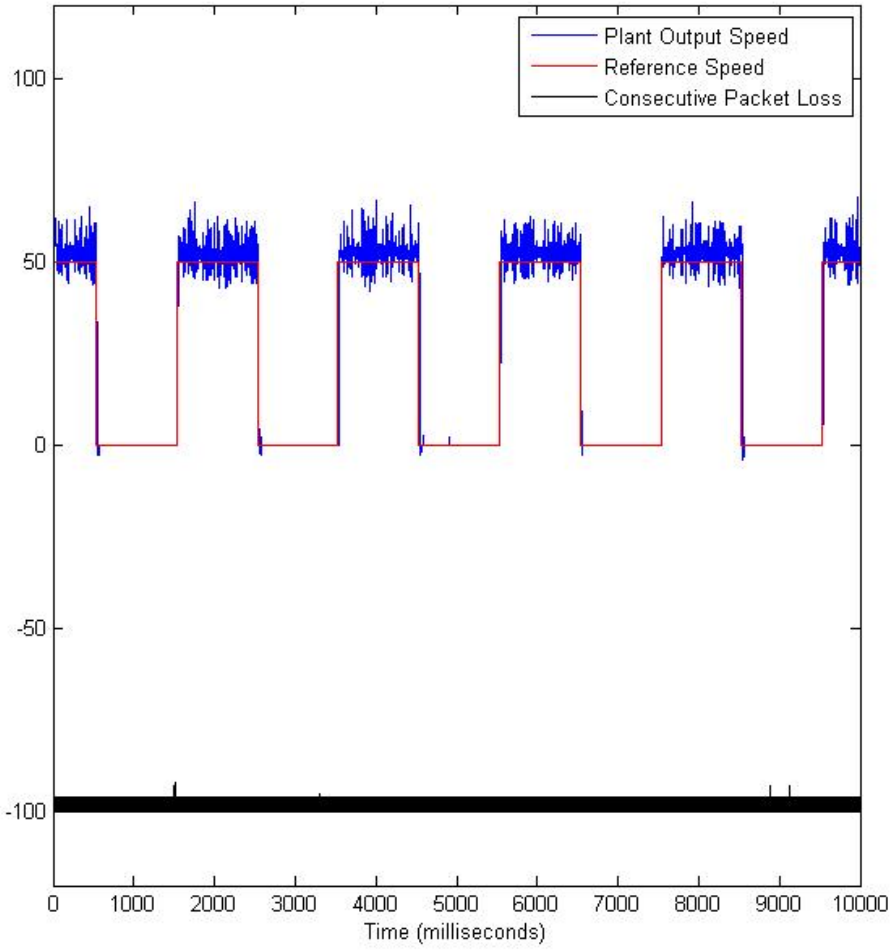

Figure 5-9 – 70 % Packet Loss – MBPNCS

RMS error of case 1.3 is 2.3269. Though we reached 70 % packet loss, as we can see from the Figure 5-9, still there is no major degradation in the stability of the system. No unstable behavior or latency in reference tracking can be observed with eyes. For step response, overshoot is increased from 16 % to 28 % in comparison with case 1.1 and undershoot is 0 %.

Figure 5-10 is the detailed view of a region from Figure 5-9 where the reference speed is changed from 0 radian/s to 50 radian/s.

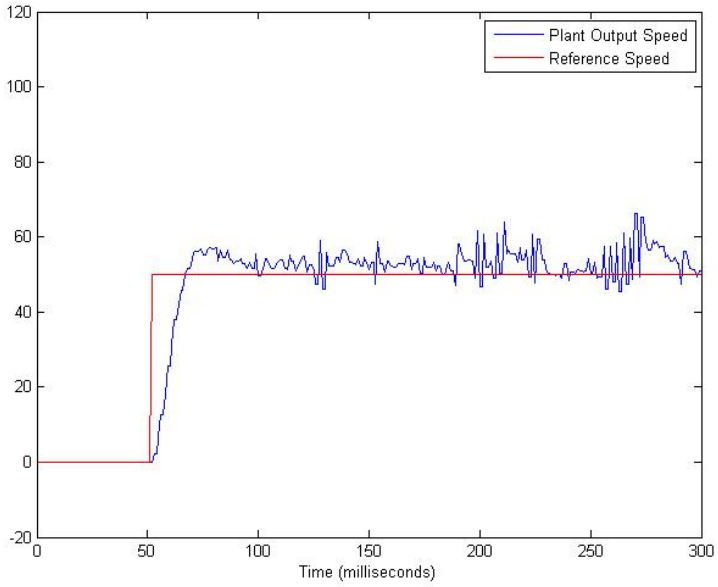

Figure 5-10 – Detailed View of a Step-Up - 70 % Packet Loss – MBPNCS

Figure 5-11 is the detailed view of a region from Figure 5-6 where the reference speed is changed from 50 radian/s to 0 radian/s.

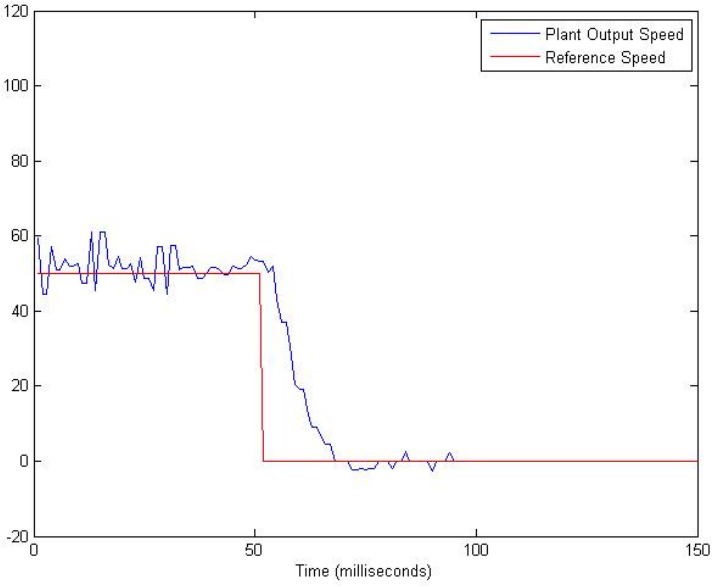

Figure 5-11 – Detailed View of a Step-Down- 70 % Packet Loss – MBPNCS

Case 1.4: 90 % packet loss

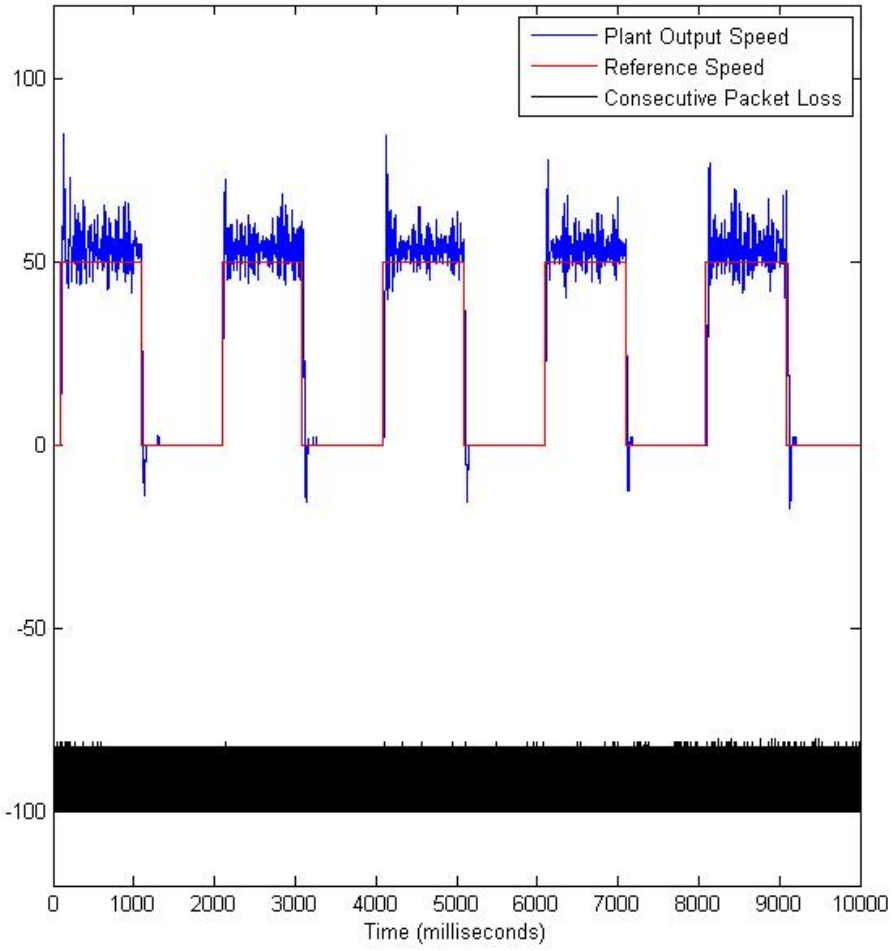

Figure 5-12 – 90 % Packet Loss – MBPNCS

RMS error of case 1.4 is 3.2109. Though still there is no major degradation in the stability of the system, for step response, overshoot is increased from 16 % to 45% in comparison with case 1.1 and undershoot is 0 %. Latency in reference tracking can now be observed with eyes.

Figure 5-13 is the detailed view of a region from Figure 5-12 where the reference speed is changed from 0 radian/s to 50 radian/s.

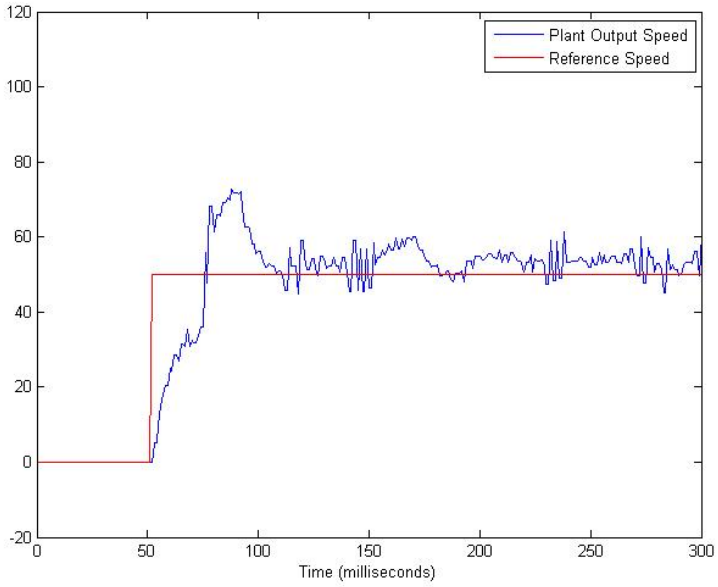

Figure 5-13 – Detailed View of a Step-Up - 90 % Packet Loss – MBPNCS

Figure 5-14 is the detailed view of a region from Figure 5-12 where the reference speed is changed from 50 radian/s to 0 radian/s.

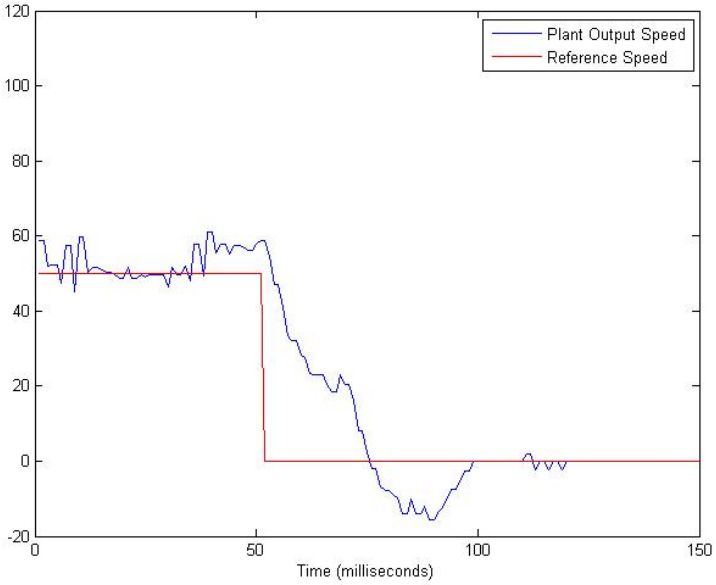

Figure 5-14 – Detailed View of a Step-Down- 90 % Packet Loss – MBPNCS

Case 1.5: 98 % packet loss

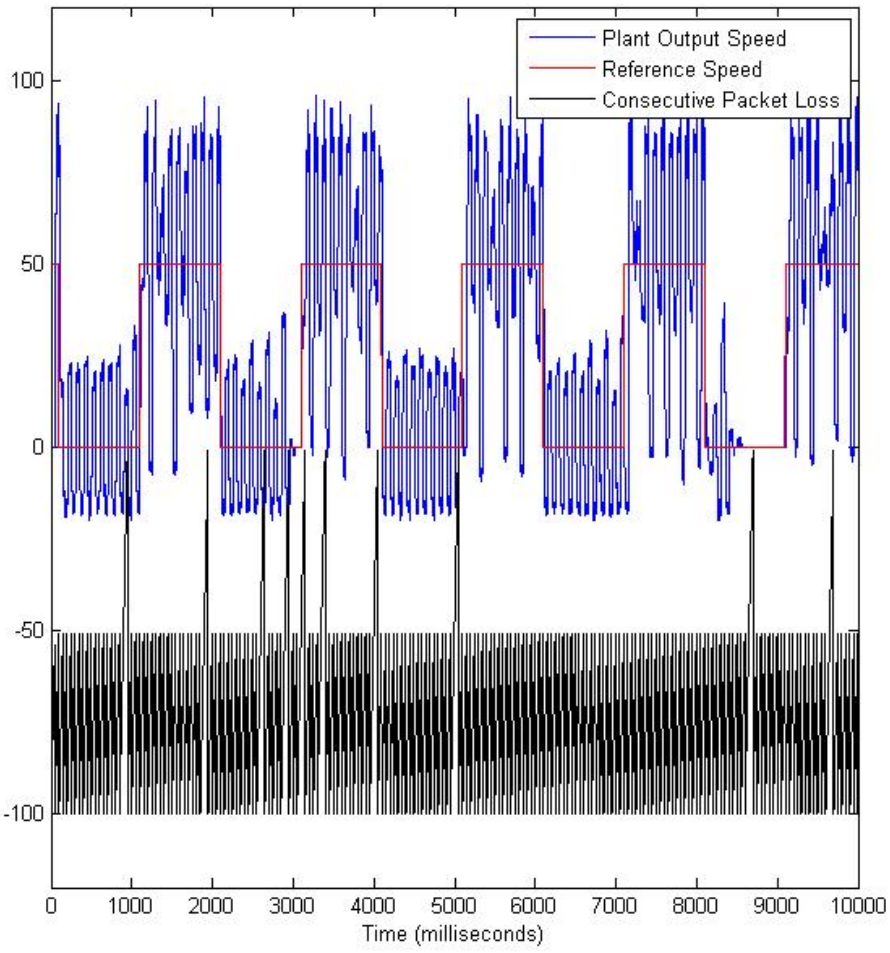

Figure 5-15 – 98 % Packet Loss – MBPNCS

RMS error of case 1.4 is 20.2437. At 98 % packet loss, the system is not stable and deviates from the reference. Instability of the system and latency in reference tracking can be easily detected with eyes. Reference is hardly followed. For step response, overshoot is increased from 16 % to 88 % and undershoot is increased from 0 % to 36 % in comparison with case 1.1.

Figure 5-16 is the detailed view of a region from Figure 5-15 where the reference speed is changed from 0 radian/s to 50 radian/s.

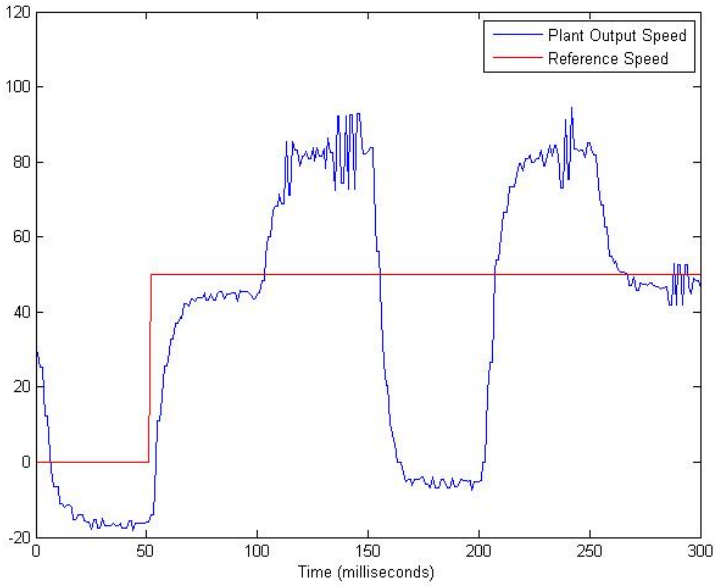

Figure 5-16 – Detailed View of a Step-Up - 98 % Packet Loss – MBPNCS

Figure 5-17 is the detailed view of a region from Figure 5-15 where the reference speed is changed from 50 radian/s to 0 radian/s.

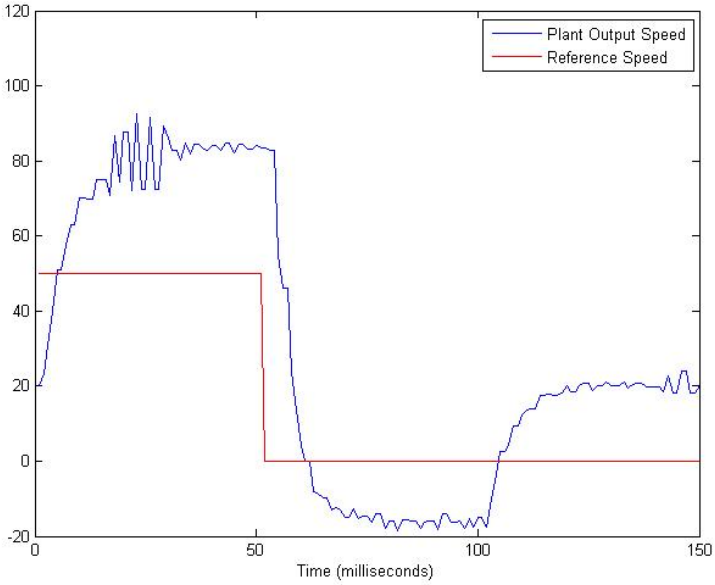

Figure 5-17 – Detailed View of a Step-Down- 98 % Packet Loss – MBPNCS

5.2.2 Category Two: Stochastic Packet Losses

In category 2, we examined two cases with 50 % and 70 % and compared the results with same cases in category 1. The aim of this category to understand if there is a change in the stability of the system in the case of stochastic packet losses.

Case 2.1: 50 % stochastic packet loss

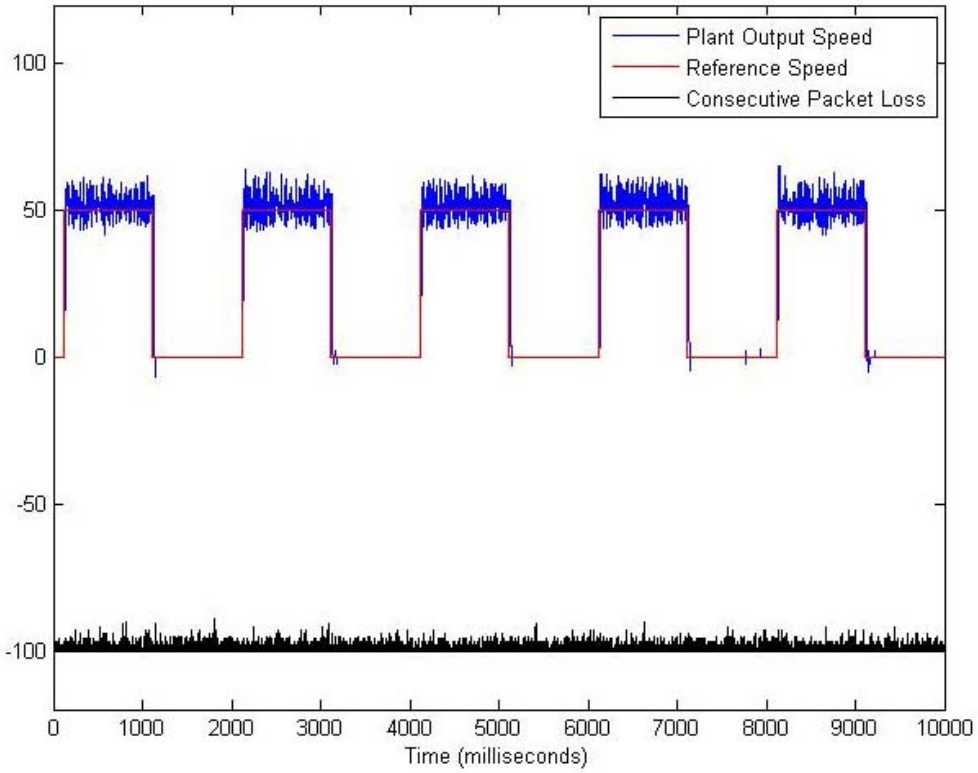

Figure 5-18 – 50 % Stochastic Packet Loss – MBPNCS

RMS error of case 2.1 is 2.0978 Gathered results are compared with case 1.2 and similar results are observed. Result shows that whether packet losses follows a pattern or not, does not have an effect on the stability of the system.

Figure 5-19 is the detailed view of a region from Figure 5-18 where the reference speed is changed from 0 radian/s to 50 radian/s.

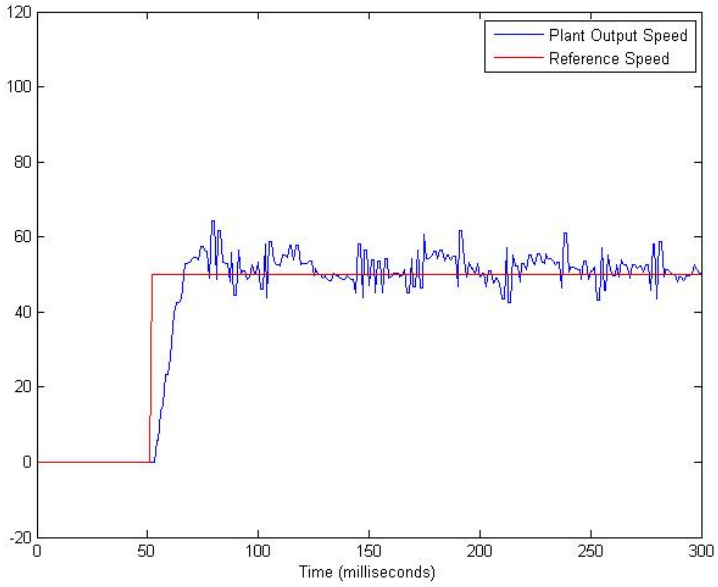

Figure 5-19 – Detailed View of a Step-Up - 50 % Stochastic Packet Loss – MBPNCS

Figure 5-20 is the detailed view of a region from Figure 5-18 where the reference speed is changed from 50 radian/s to 0 radian/s.

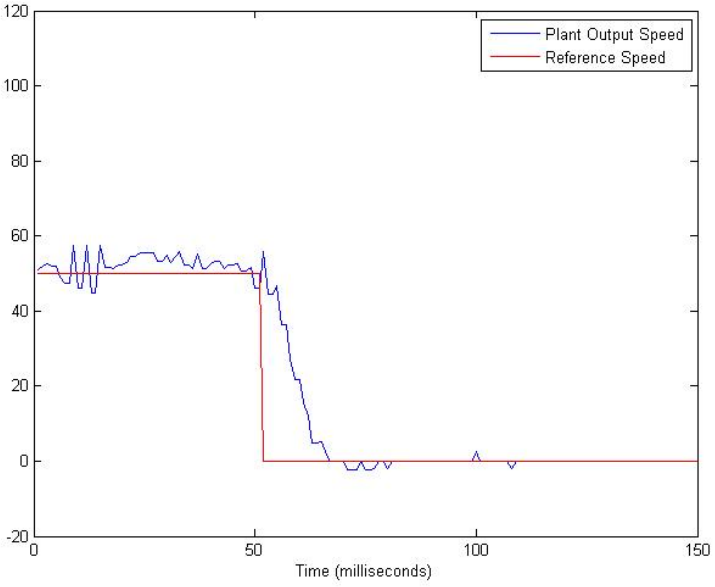

Figure 5-20 – Detailed View of a Step-Down- 50 % Stochastic Packet Loss – MBPNCS

Case 2.2: 70 % stochastic packet loss

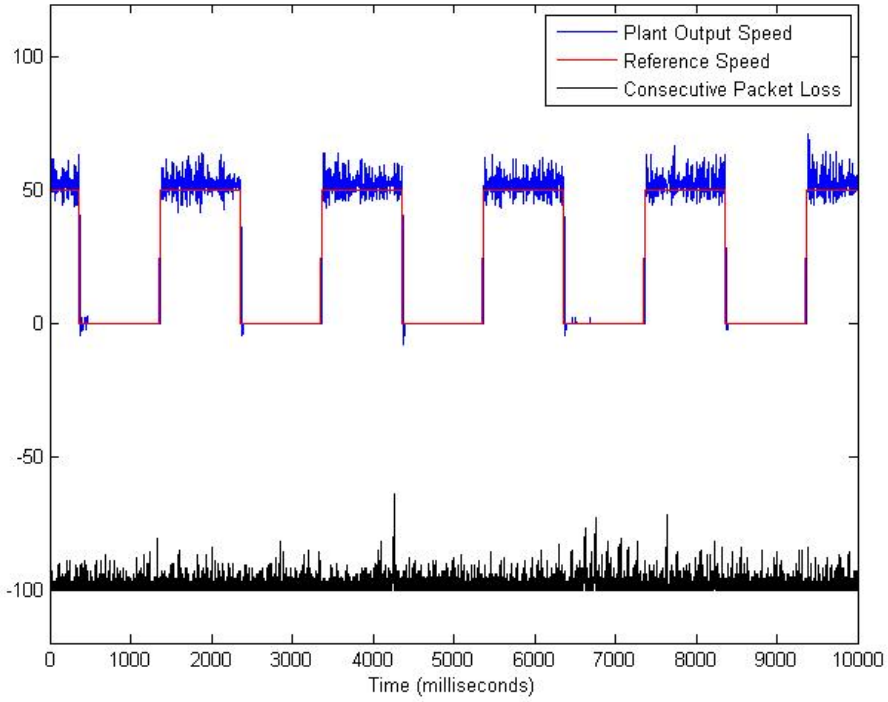

Figure 5-21 – 70 % Stochastic Packet Loss – MBPNCS

RMS error of case 2.2 is 2.3789. Gathered results are compared with case 1.3 and similar results are observed. Result shows that whether packet losses follows a pattern or not, does not have an effect on the stability of the system.

Figure 5-22 is the detailed view of a region from Figure 5-21 where the reference speed is changed from 0 radian/s to 50 radian/s.

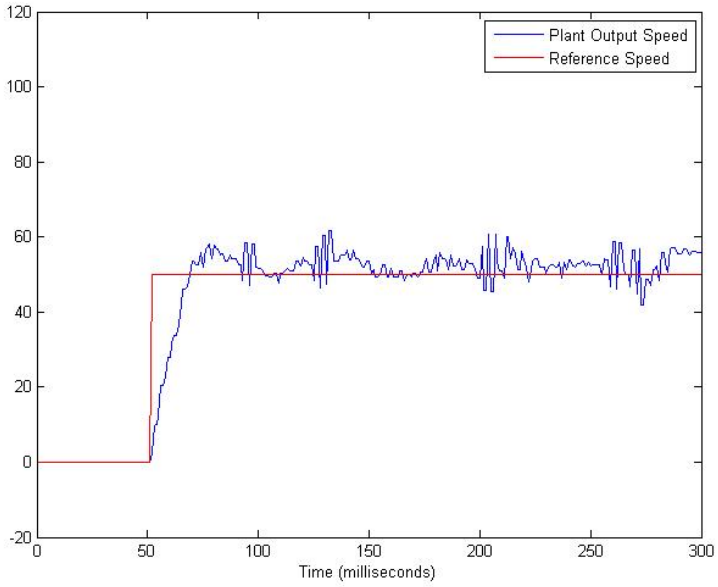

Figure 5-22 – Detailed View of a Step-Up - 70 % Stochastic Packet Loss – MBPNCS

Figure 5-23 is the detailed view of a region from Figure 5-21 where the reference speed is changed from 50 radian/s to 0 radian/s.

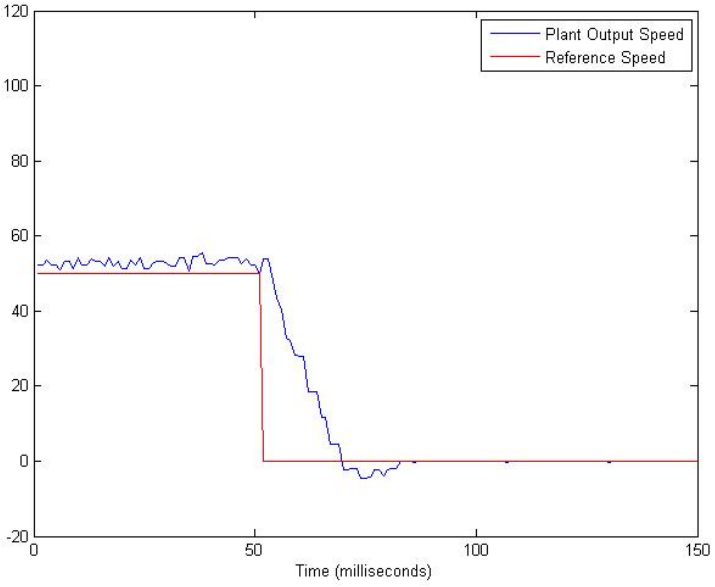

Figure 5-23 – Detailed View of a Step-Down- 70 % Stochastic Packet Loss – MBPNCS

6 CONCLUSION

In this thesis, we aimed to make an implementation of the MBPNCS by preparing a highly scalable and reliable NCS.

Software Work

- A Linux distribution from scratch was prepared for the embedded computer system to run.
- Later, the Linux kernel was patched and compiled again for RT Linux support.
- In order to add networking support to real-time processes, RTSock framework was installed.
- Device drivers were prepared for ADC, DAC and encoder reader
- To gather data from the controlled system, a user space process was prepared that communicates with the kernel modules via RT-FIFO's.
- Real time processes were prepared for each node executing the base duties of the nodes
- Various performance tester programs were prepared to observe jitter, minimum possible sampling time of the PC104's and utilization of the network

Hardware Work

- Most of the devices were purchased from various vendors
- Intermediate circuits were prepared for interoperability of the devices
- Platform assembling and cabling works were carried out

Experimental Work

- Model of the plant was prepared using parametric methods, later optimizing using the hill-climbing method

- Kernel modules of the nodes were altered such that random packet losses become possible with a desired rate.

Obtained results showed that MBPNCS method can sustain the stability of the controlled system even in the case of high stochastic or non-stochastic packet losses. In the case that temporary packet loss continues till the stack of predicted control output signals are emptied in the actuator node, last predicted control output signal is continuously applied to the plant till new packet arrives. While the stack stays empty, stability of the system is uncontrollable and plant dependent. Empty stack problem should be considered as a future work.

The range for the type of communication mediums to be used in an NCS has been increased with the implemented method. Even the Internet network can be used with a best effort service level agreement when long-distance control system mechanisms are needed. On the other hand, majority of the real life control mechanisms require a distributed network in a limited area with an average size of a factory. For such areas, establishing a dedicated best effort Ethernet network is not costly. However, establishing a real-time network that guarantees packet transmission in the given sub milliseconds is highly costly in comparison to best effort Ethernet network. For this reason, for NCS's distributed in a limited factory like area, implemented method is also favorable.

On the other hand, practical issues bound the fault tolerance of a control system. Since we assume that the communication medium is unreliable, it may be a good idea to define the unreliability of the network using mathematical methods. Effect of the load of model packets on the complexity of communication medium should be investigated.

In this work, design and programming has been prepared such that, adding multiple sensors or actuators can be done by a simple change in the configuration file. This design decision allows us to easily add multiple plants to the system for future work.

REFERENCES

[1] A.T. Naskali, A. Onat, "Model Based Predictive Networked Control Systems", *Ms. Thesis, Sabancı University*, 2006

[2] A.T. Naskali, A. Onat, "Random Network Delay in Model Based Predictive Networked Control Systems", *WSEAS Intl. Conf. on AppliedComputer Science, pp 199-206*, 2006

[3] M. Colnaric "Design of Embedded Control Systems" *ICIT 2003* Maribor, Slovenia 2003

[4] M.S. Branicky, S.M. Phillips, Wei Zhang, "Scheduling and feedback co-design for networked control systems," *Proc. 41st IEEE. Conf. on Decision and Control*, vol.2, no.pp. 1211- 1217 vol.2, 10-13 Dec. 2002

[5] X. Liu and A. J. Goldsmith, "Wireless Medium Access Control in Networked Control Systems," *IEEE American Control Conference*, 2004

[6] X. Liu and A. J. Goldsmith, "Wireless Network Design for Distributed Control" *IEEE Conference on Decision and Control*, 2004

[7] J. Yook, D. Tilbury; N. Soparkar "Performance Evaluation of Distributed Control Systems With Reduced Communications" *IEEE Control Systems Magazine*, Vol. 21, no. 1, pp. 84-99, 2001.

[8] P Otanez; J. Moyne, D. Tilbury "Using Deadbands to Reduce communication in Networked Control Systems" *Proceedings of the 2002 American Control Conference*. 2002

[9] 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" *Proceedings of 2000JUSFA 2000 Japan-USA Symposium on Flexible Automation* July 23-26, 200, Ann Arbor, Michigan, USA 2000

[10] K. Natori, K. Onishi "An approach to design of feedback systems with time delay", *Industrial Electronics Society, 2005. IECON 2005. 32nd Annual Conference of IEEE 6-10 Page(s):6 pp*, Nov. 2005

[11] O. J. M. Smith, "A controller to overcome dead time," *ISA Journal, vol. 6, no. 2, pp. 28–33*, 1959.

[12] JB Rawlings, “Tutorial Overview of Model Predictive Control”, *IEEE Control Systems Magazine*, Vol. 20, No. 3, June 2000, pages 38-52.

[13] FSM Labs, The RTCore Index – Accounting Performance, http://www.altosys.co.in/pdfs/RTLinuxPro%20Performance.pdf, 2004

[14] Iperf, TCP/UDP Bandwidth Measurement Tool, Version 1.7.0, http://dast.nlanr.net/Projects/Iperf/

[15] C. Hornig, " A Standard for the Transmission of IP Datagrams over Ethernet Networks", *RFC-879*, April 1984.

[16] D. Borman, “TCP and UDP over IPv6 Jumbograms”, *RFC-2147*, May 1997

[17] Kontron , “104-ADIO128 User’s Guide”, November 2002