DELAY MODELING AND CONTROLLER DESIGN FOR NETWORKED CONTROL SYSTEMS

BY

Jimmy Kin Cheong So

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science,
Graduate Department of Electrical and Computer Engineering,
in the University of Toronto.

For my family.

Delay Modeling And Controller Design For Networked Control Systems

Master of Applied Science Thesis
Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto

by Jimmy Kin Cheong So
September 2003

Abstract

This thesis discusses modeling and control issues for Networked Control Systems (NCS). NCS is a feedback control system wherein the control loops are closed through a real-time network. Such systems offer advantages such as lower installation costs, increased flexibility, and rapid installation. However, communication delay is a general problem in NCS, and it could destabilize the closed loop system. In the first half of this thesis, we model the delay for three popular networks for control applications: Controller Area Network (CAN), Process Field Bus (Profibus), and fully-switched Ethernet. In the second half, control strategies are provided for control systems with constant or random delay. Systems with constant delay can be controlled by modified Smith predictor; systems with random delay can be controlled by dynamic controller. The results of the thesis can be used to provide guidance on the choice of network and controller design in a networked control system environment.

Acknowledgements

I would like to thank my supervisor Professor Raymond H. Kwong for his advice and guidance, as well as his financial support. I would like to thank K. I. Choi for proof reading the manuscript. I would like to thank my family and friends for their continuing support and care. I would like to acknowledge financial support from the University of Toronto.

Contents

List of Figures

Chapter 1

Introduction

As the price and performance of digital computers continue to improve and their size, weight, and power requirements continue to decrease, there has been a steady increase in the use of computer-based real-time systems in a wide variety of fields. Application domains such as military, industry, and medicine indicate a wide spectrum of possible implementations. Current real-time system examples include nuclear power plant control, industrial manufacturing control, medical monitoring, space navigation and guidance, and weapon delivery systems. By taking advantage of computer control, we are able to control more complex system. However, such a complex system will involve a lot of sensors and actuators. The typical point-to-point wiring scheme will make installation and maintenance more difficult. Therefore, a new type of wiring scheme has emerged. It is to connect sensors, actuators, and processors through a network.

Feedback control systems wherein the control loops are closed through a real-time network are called networked control systems (NCS) [25, 47]. The defining feature of an NCS is the information (reference input, plant output, control input, etc.) is exchanged using a network connecting control system components (sensors, controller, actuators, etc.). Fig. 1.1 illustrates a typical setup and the information flow of an

NCS. Such systems offer a number of advantages over their traditional predecessors, including lower installation and maintenance costs, increased flexibility and rapid installation and diagnostics.

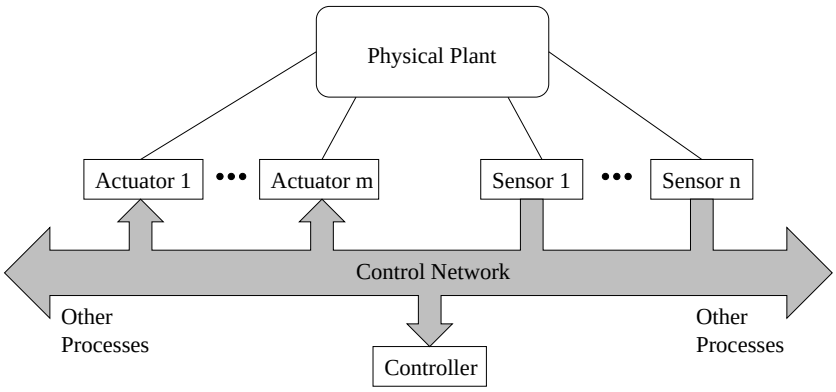


Figure 1.1: A typical NCS setup and information flows

The insertion of the communication network in the feedback control loop makes the analysis and design of an NCS complex. Conventional control theories with many ideal assumptions, such as non-delayed sensing and actuation, must be re-evaluated before they can be applied to NCSs. Network-induced delay (sensor-to-controller delay and controller-to actuator delay) occurs while exchanging data among devices connected to the shared medium. This delay, either constant or time varying, can degrade the performance of control systems designed without considering the delay and can even destabilize the system. In [17], the authors have considered network delay in the feedback loop. However, they do not provide a mathematical model for the network delay and do not show how to calculate this delay factor from a specific network. Furthermore, the controller design is only applicable for system with constant delay. Similarly, in [36], authors did not provide any guidance on calculating network delay and the controller is not applicable to open-loop unstable system. In [26], authors have constructed a simulator to model network delay that analyze closed loop system performance. However, they did not provide any guidance

on choosing network and control strategy.

In 1957, O.J.M. Smith [33, 19] presented a control scheme for single-input single-output (SISO) systems, which has the potential of improving the control of loops with time delay. This scheme became known as the Smith predictor. However, a main drawback, linked to internal instability of the prediction, is that it will fail to stabilize an unstable system. In [20, 21], the authors have developed a predictor that is similar to Smith predictor that can stabilize unstable system. This controller is easy to design and can stabilize a system with constant delay. However, it requires full observation of system's internal states and the stability analysis is not applicable to system with random delay.

In 2000, Lin Xiao, Arash Hassibi, and Jonathan How [45] presented an controller design procedure for system with random delay. They model the random delay as a Markovian process [25, 45]. The resulting controller can make the closed loop system with a transition probability matrix mean square stable [6]. Although the controller design procedure is easy to follow, it is quiet demanding on computation power. In some cases, it will take more than 48 hours to do the controller design and this controller design strategy is not suitable for constant delay.

From control perspective, control network designer will focus on two issues: type of delay and reliability. In the last two paragraphs, we have discussed controller design strategy for system with constant or random delay. Since each controller design has advantages on system with different type of delay, it is important to take advantage on this by choosing an appropriate controller design strategy for different networks. Also, different networks will have different reliability, such as: error detection and error correction. Of course, the reliability issues depend on the quality of services that the system wants to provide. Since all of these issues are network dependent, it is important to understand the network behavior. Therefore, in this thesis, we will

analysis the timing behavior of three popular networks in control industry: Controller Area Network (CAN), Process Field Bus (Profibus), and Ethernet.

For CAN and Profibus, detailed timing analyses have been studied in [12, 38] and [22, 40], respectively. However, their models are incomplete because they did not characterize transmission error appropriately. It is important to include transmission error to the timing model because it is often unavoidable and it will substantially increase the response time. By including transmission error to our model, we are able to provide a more realistic estimate of the response time. For Ethernet, we will introduce our network configuration model with timing analysis, which is much suitable for control application as compare to the traditional network configuration.

Using the timing analysis, we have characterized delay models for CAN, Profibus, and Ethernet. The models provide information about delay characteristics, such as range of delay and randomness of delay. We then use this information to choose our controller design strategy. There are two type of controllers studied in this thesis. These controllers, the resetting Smith predictor and a dynamic controller based on a Markov jump system model, can stabilize system with constant or random delay, respectively. From our analysis, we find that systems implemented through CAN and Profibus experience constant delay. Hence, we suggest using the resetting Smith predictor to contol the system. On the other hand, we find that systems implemented through Ethernet experience random delay. As a result, we suggest using the dynamic controller based on a Markov jump system model to control the system. This provides guidance on how to choose and design controller for networked control systems.

We now give an outline of the thesis. In Chapter 2, we present the protocol description and timing analysis of Controller Area Network. CAN's timing analysis have been studied in [12, 38]. However, they characterize the error function as a function of current message only. It is unrealistic to consider current message retransmission

because CAN is a priority based network so that high priority message retransmission will directly affect the response time for lower priority message. Our model will consider the above situation. At the end of this chapter, we will discuss this network from the control perspective.

In Chapter 3, we present the protocol description and timing analysis of Process Field Bus. Profibus's timing analysis have been studied in [22, 40]. However, they did not consider the possibility of transmission error that leads to message retransmission. Our model includes message retransmission. At the end of this chapter, we discuss this network from the control perspective.

In Chapter 4, we first present the protocol description of Ethernet. Secondly, we introduce the traditional network configuration and explain why this configuration is not suitable for control application. Thirdly, we introduce our network configuration, fully switched Ethernet network with full-duplex cables, which is suitable for control application. Finally, we go over the timing analysis for this network configuration with an example at the end.

In Chapter 5, we discuss controller design for networked control system. In section 5.1, we state the problem and major assumptions for systems with input and output delay. In section 5.3, we discuss a modified Smith predictor that has been studied in [20, 21] for systems with constant delay. At the end of that section, we go through the design procedure by using a magnetically levitated ball as an example. In section 5.5, we discuss controller design strategy that has been studied in [45] for systems with random delay. At the end of that section, we demonstrate the design procedure by using the magnetically levitated ball system.

In Chapter 6, we summarize the key features and limitations for CAN, Profibus, and Ethernet. Also, we summarize our discussion on controller design for system with constant or random delay. Furthermore, the requirements for using each controller

will be discussed. Finally, we discuss the direction of future research on this topic.

Chapter 2

Controller Area Network

Controller Area Network (CAN) was first developed by Robert Bosch GmbH, Germany in 1986, as a communication system between three ECUs (electronic control units) in vehicles by Mercedes. They found that an UART[1] is no longer suitable in this situation because it is only capable of point-to-point communication. If all possible combinations of switches, sensors, motors, and other electrical devices in fully featured vehicles are accumulated, the resulting number of connections and dedicated wiring is enormous. Networking provides a more efficient method for today's more complex in-vehicle communication. CAN is a serial network, established among microcontrollers, that is primarily used in embedded systems. It is a two-wire, half duplex, high-speed network system and is well suited for high speed applications using short messages. CAN can theoretically link up to 2032 devices on a single network. However, due to the practical limitation of the hardware (transceivers), it can only link up to 110 nodes on a single network. It offers high-speed communication rate up to 1 Mbits/sec that allows real-time control. In addition, the error confinement and the error detection feature make it more reliable than other networks in noise critical

[1]UART (Universal Asynchronous Receiver/Transmitter) is the microchip with programming that controls a computer's interface to its attached serial devices.

environment.

In section 2.1, we describe the CAN protocol and its way of handling transmission errors. Section 2.2 is devoted to calculating the worst case response time under noise critical environment for an industrial example. Finally, in section 2.3, we summarize CAN's features.

2.1 Protocol Description

This section will discuss some key features in the CAN protocol. It is important to understand how the CAN protocol works because this will enable us to correctly formulate its timing behavior that is crucial to control applications. Later in this section, we will discuss CAN's error handling and recovery process. It will not only affect the timing behavior but also affect the decision making on choosing this network for a particular application.

2.1.1 Detection Method and Identifier

CAN is a multi-master network that can be composed of many controllers who share different controls and tasks. The master node can actively invoke a slave node to provide data that the master node requires and it can also send information to other nodes. CAN is similar to Ethernet in that the CAN node checks if the bus is busy before sending a message and it uses collision detection method. However, when an Ethernet network detects a collision, both sending nodes stop transmitting (destructive collision detection mechanism) and wait a random time before trying to send again. Ethernet networks are very sensitive to high bus loads due to such collision handing. On the other hand, when a CAN network detects collision, the high priority node will continue sending data while the low priority node will stop sending data and

start listening to the bus (constructive collision detection mechanism). This arbitration is called CSMA/CR (Carrier Sense, Multiple Access with Collision Resolution).

Data messages transmitted from any node on a CAN bus do not contain any addresses of either transmitting node, or of any intended receiving node. Instead, the message is labeled by an identifier that is unique throughout the network. All other nodes on the network receive the message and each performs an acceptance test on the identifier to determine if the message, and thus its content, is relevant to that particular node. The message will be processed if it is relevant; otherwise it is ignored.

The unique identifier identifies the type of message and the message priority. The lower the numerical value of the identifier, the higher the priority. This allows arbitration if two (or more) nodes compete for access to the bus at the same time. The higher priority message is guaranteed to gain bus access as if it were the only message being transmitted. Lower priority messages are automatically re-transmitted in the next bus cycle, or in a subsequent bus cycle if there are still other higher priority messages waiting. Each CAN message has an identifier which is 11 bits (CAN Version 2.0A) or 29 bits (CAN Version 2.0B). This identifier is located in the beginning of each CAN message. A transmitting node always listens on the bus while transmitting. A node that sends a high in the arbitration field and detects a low would realize that it has lost arbitration. As a result, it stops transmitting and lets the other node with a higher priority message continue uninterrupted.

2.1.2 CAN Frames

There are two kind of frames in CAN: data frames and remote frames. Data frames are used when a node wants to transmit data on the network. Remote frames can be described as a request for information. A frame with the Remote Transmit Request

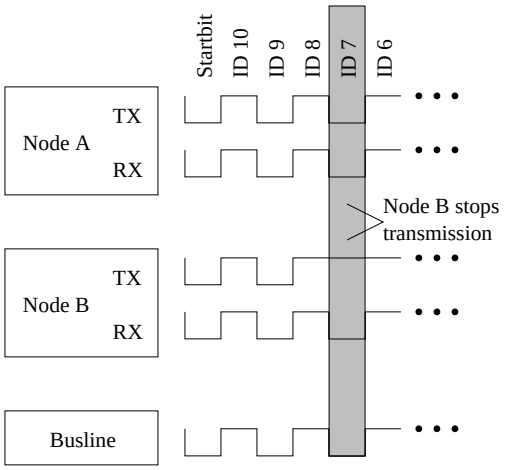


Figure 2.1: CAN arbitration

(RTR) bit set means that the transmitting node is asking for information of the type given by the identifier. A node which has the available information should then respond by sending the information onto the network. The CAN protocol[2] supports two message frame formats that are standard CAN (Version 2.0A) and extended CAN (Version 2.0B). It is important to know that most 2.0A controllers can transmit and receive only standard format messages, while 2.0B controllers can handle both type of messages.

CAN standard frame

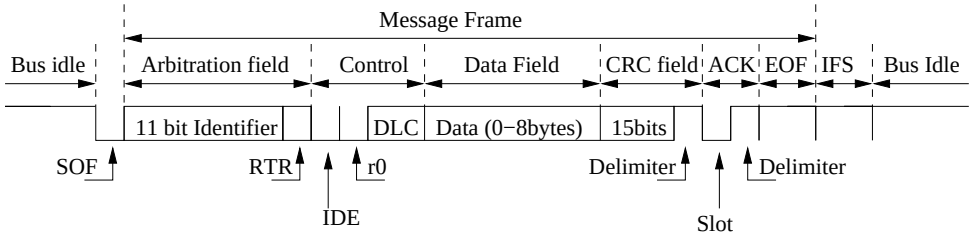


Figure 2.2: CAN standard frame [15]

[2]The CAN protocol is an international standard defined in the ISO 11898

A message in the CAN standard frame format begins with the start bit called Start Of Frame (SOF), this is followed by the Arbitration field which consists of the identifier and the Remote Transmission Request (RTR) bit used to distinguish between the data frame and the remote frame. The following Control field contains the Identifier Extension (IDE) bit to distinguish between standard frame and extended frame, as well as the Data Length Code (DLC) used to indicate the number of following data bytes in the Data field. If the message is used as a remote frame, the DLC contains the number of requested data byte. The Data field that follows is able to hold up to 8 data byte. In addition, a reserved bit (r0) is included in the control field for future extension. The integrity of frame is guaranteed by the following Cyclic Redundant Check (CRC) sum. The Acknowledge (ACK) field compromises the ACK slot and the ACK delimiter. The bit in the ACK slot is sent as a recessive bit and is overwritten as a dominant bit by those receivers which have at this time received the data correctly. Correct messages are acknowledged by the receivers regardless of the result of the acceptance test. The end of the message is indicated by End Of Frame (EOF). The Intermission Frame Space (IFS) is the minimum number of bits (7 bits) separating consecutive messages. If there is no following bus access by any station the bus remains idle.

CAN extended frame

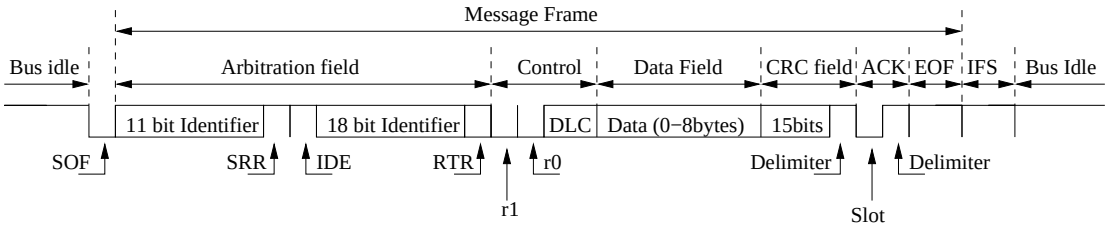


Figure 2.3: CAN extended frame [15]

As stated in the beginning of this chapter, CAN is developed in Germany and soon become a standard in Europe. However, this standard frame format is not compatible to the USA standard. As a result, the extended frame format evolved to provide compatibility with other serial communications protocols used in automotive applications in the USA. Ver. 2.0B's Arbitration field contains two identifier bit fields: the first (base ID) is eleven bits long for compatibility with Ver. 2.0A and the second field (ID extension) is eighteen bits long, giving a total length of twenty nine bits.

Substitute Remote Request (SRR) bit is included in the Arbitration Field. The SRR bit is always transmitted as a recessive bit to ensure that, in the case of arbitration between a Standard Data Frame and an Extended Data Frame, the Standard Data Frame will always have higher priority if both messages have the same base (11 bit) identifier.

Since the identifier of the standard frame is 11 bit wide, 2048 different message types are possible. Due to implementation reasons only 2032 of them can be used. However, the 29 bit identifier of the extended frame supports more than 500 million different message types. As CAN can only linked up to 110 nodes. there is no advantage of using extended frame in practice. Extended frame not only provide no practical advantage over standard frame, but also has numerous disadvantages as compare to standard frame. In [30], it was shown that the extended frame had higher bus access time, lower bus throughput and higher cost. In addition, one can expect that extended format will have higher bit error rate than standard format because both formats have the same number of CRC bit. Because of these numerous disadvantages, standard frame is the most common format used in the CAN industry.

2.1.3 Error Process and Detection

In [42], the authors have estimated the expected number of undetected transmission errors during lifetime of a vehicle is lower than 10^{-12}. Such an excellent performance is the result of a very efficient error detection mechanism being used in CAN. This mechanism can be divided into message level and bit level. In message level, CRC safeguards the information in the frame by adding redundant check bits at the transmission end. At the receiver these bits are re-computed and tested against the received bits. If they do not agree, there has been a CRC error. Frame check verifies the structure of the transmitted frame by checking the bit fields against the fixed format and the frame size. Errors detected by frame checks are designated format errors. Finally, frames received are acknowledged by all receivers through positive acknowledgement. If no acknowledgement is received by the transmitter of the message an ACK error is indicated. At the bit level, the transmitter monitors the bus signals and detects errors. Each transmitting station observes signal on the bus and thus detects differences between the bit sent and the bit received. This permits reliable detection of global errors and errors local to the transmitter. The coding of the individual bits is tested at bit level. Each bit is represented by Non Return to Zero (NRZ) coding [16], which guarantees maximum coding efficiency. The synchronization edges are generated by bit stuffing; that means after five consecutive equal bits, the transmitter inserts into the bit stream a stuff bit with the complementary value, which will then be removed by the receivers. The code check is limited to checking adherence to the stuffing rule. If one or more errors are discovered by at least one station using the above mechanisms, the current transmission is aborted by sending an "error flag". This prevents other stations accepting the message and thus ensures the consistency of data throughout the network. After transmission of an erroneous message has been aborted, the sender automatically re-attempts transmission. If one

or more errors are discovered by at least one station using the above mechanisms, the current transmission is aborted by sending an error flag. This prevents other stations from accepting the message and thus ensures the consistency of data throughout the network. After transmission of an erroneous message has been aborted, the sender automatically re-attempts transmission. There may again be competition for bus allocation.

However, in the event of a defective station, it might lead to all messages (including the correct ones) being aborted. If no self-monitoring measure were taken, the bus system would be blocked by this. The CAN protocol therefore provides a mechanism to distinguish sporadic errors from permanent errors and local failures at the station. This is done by statistical assessment of station error situations, with the aim of recognizing a station's own defects and possibly entering an operation mode (normal mode and bus off mode[15]), where the rest of the CAN network is not negatively affected. This may go as far as the station switching itself off to prevent messages from being erroneously recognized as incorrect.

2.2 Timing Analysis

In the past, the approaches to real time on CAN was very passive. People will throw all the messages at the CAN bus and only worry about the timing behavior when problems start to show up in testing. This generally works with a low bus utilization because the chance of a long delay is fairly small. However, if people want to fully utilize the existing CAN, then they will have to know that the timing requirements of the application will be met. This motivates researchers [38, 37, 39] from the University of York in England to formulate the first mathematical model that describes the timing behavior of CAN and similarly in [12]. The next section will summarize their results and discuss the incompleteness in their model.

2.2.1 Literature Review

A periodic frame m is characterized by (C_m, T_m, D_m, S_m) where T_m is the period, C_m the transmission time, D_m the deadline and S_m the data size in bytes. A message m can be delayed by higher priority messages and by a lower priority message that has already begun transmission. The longest time that a low priority message, lower than m, can possibly hold the bus is denoted as B_m. The worst-case response time for a message m under the assumption of no hardware failure is given as follow:

$$R_m = t_m + C_m \tag{2.1}$$

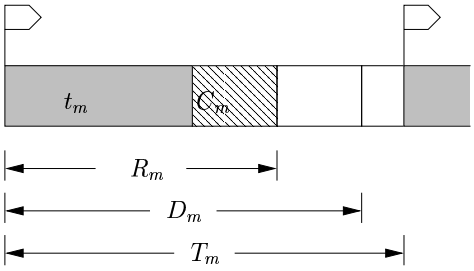


Figure 2.4: Timing Diagram

The queuing time, t_m, is measured from when the message is queued to the start of winning arbitration. The transmission time, C_m, is the time taken to actually send the message on the bus. This transmission time can be found from the maximum size of the message, and by knowing the maximum number of stuff bits that can be inserted into the bit stream when transmitting the message. The following equation gives the transmission time C_m[15]:

$$C_m = \left(\left\lfloor \frac{34 + 8S_m}{4} \right\rfloor + 47 + 8S_m \right) \tau_{bit} \tag{2.2}$$

where 47 is the size of the fixed-form bit fields of the CAN standard frame, $8S_m$ is

the data size in bits, and τ_{bit} is the bit time. The floor function[3] correspond to bit stuffing as mentioned in Section 2.1.3 and out of the 47 overhead bit, 34 are subject to bit-stuffing[37]. Notice how the divisor in the equation is 4 and not 5, even though CAN uses a stuff width of 5. This is because stuff bits are themselves subject to bit stuffing. Consider the following unstuffed bit stream:

$$\ldots 000011110000111100001111\ldots$$

If a '0' stuff bit is inserted at the start of the stream, then the final stream becomes:

$$\ldots 0000\mathbf{1}1111\mathbf{0}0000\mathbf{1}1111\mathbf{0}0000\mathbf{1}1111\ldots$$

The inserted stuff bits are marked in bold. Therefore, the worst case would be an extra bit inserts for every four unstuffed bit instead of five unstuffed bits. A periodic frame m and its worst case response time are shown in Fig. 2.4 where the "flag" corresponding to the release time of message.

In order to determine the maximum queuing time that message m will experience, we need to know the longest time that a low priority message can possibly hold the bus before arbitration starts. This is equal to the maximum transmission time of the largest message of lower priority, Cl_{max}. In addition, we need to calculate the maximum time that all messages with higher priority than m to transmit while message m still in queue. This latter time is given by:

$$\sum_{\forall j \in hp(m)} \left\lceil \frac{t_m + \tau_{bit}}{T_j} \right\rceil C_j \tag{2.3}$$

where $hp(m)$ is the set of messages of higher priority than message m, t_m is the longest

[3] $\lfloor a \rfloor = b$ where b is the biggest integer smaller than or equal to a

time that message m is queued before winning arbitration. The ceiling function[4] represents the number of rounds that message $j \in hp(m)$ transmit during the time where message m wait for transmission.

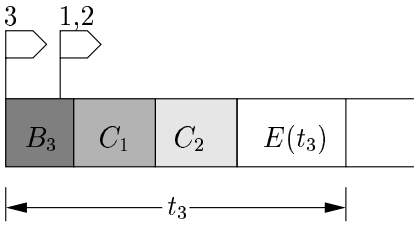


Figure 2.5: Queuing Time

Finally, the time take due to transmitting error frames and retransmitting higher priority messages needs to take into account. $E(t)$ is the "error recovery function", which is the maximum expected overhead due to errors and retransmissions on the bus in an interval t, as defined in [37]. Putting these times together we get the maximum queuing time:

$$t_m = B_m + \sum_{\forall j \in hp(m)} \left\lceil \frac{t_m + \tau_{bit}}{T_j} \right\rceil C_j + E(t_m) \tag{2.4}$$

This equation can be solved by recursive relation. An example is given in Fig. 2.5 where a lower number indicates a message with higher priority and the "flag" correspond to the instant where message start pending. However, in [24], the author characterize the "error recovery function" as a function of number of retransmission of the current message. They did not consider the effect of retransmission for higher priority which will largely increase the response time for lower priority message. Also, in [12], the author completely ignore the possibility of retransmission.

[4] $\lceil a \rceil = b$ where b is the smallest integer bigger than or equal to a

2.2.2 Improved Model

In reality, CAN is being used to connect circuits and mechanical parts. These electromechanical parts may induce electromagnetic field that corrupt signal on the bus. As a result, an improved model that incorporate retransmission and error frame transmission will be presented in this section. We will include n_m, the number of retransmissions, as another variable that characterizes the periodic frame m. Once an error detected, an error frame (EF) will be sent to the transmitter as it was described in Section 2.1.3. The recovery time from detecting an error until the possible start of next frame, is typically 17 bit times to 23 bit times (in the case of a heavily disturbed bus up to 29 bit times) if there are no further errors. For our purpose, we will assume EF has the size of 23 bit times. It is important to know that the error frame will not be interrupted by any other message. Therefore, the longest time that a low priority message can possible hold the bus should be equal to the maximum transmission of the largest message of lower priority with error. The improved model of worst case response time is defined as follow:

$$R_m = C_m + \sum_{i=0}^{n_m} t_{m_i} \tag{2.5}$$

where t_{m_i} is the time between message m's i^{th} retransmission and $i-1^{th}$ retransmission.

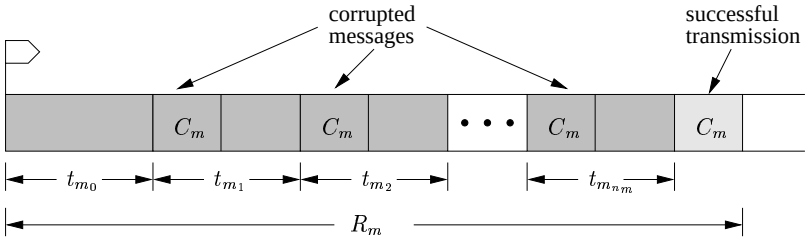


Figure 2.6: Improved Model

In order to determine the maximum queuing time that message m will experience,

we need to know the longest time that a low priority message can possibly hold the bus before arbitration starts. This is equal to the maximum transmission time of the largest message of lower priority with error. If the low priority message is corrupted, then the receiver requires to send an error frame that acknowledge the sender's message is corrupted. However, retransmission can be proceed only when no high priority message is pending. Also, we will introduce another variable O_m that is the amount of time during which message m occupies the bus that includes error frame transmission. For example, if message m has retransmitted n_m times before a successful transmission, then the time that message m has occupied the bus is $O_m = n_m \times (C_m + EF) + C_m$. The last term corresponds to successful transmission. In addition, we need to calculate the maximum transmission time that all higher priority messages (with or without error) than m while message m still in queue. This latter time is given by:

For $i = 0$,

$$\sum_{\forall j \in hp(m)} \left\lceil \frac{t_{m_i} + \tau_{bit}}{T_j} \right\rceil O_j$$

otherwise,

$$\sum_{\forall j \in hp(m)} \left(\left\lceil \frac{\sum_{p=0}^{i} t_{m_p} + \tau_{bit}}{T_j} \right\rceil - \left\lceil \frac{\sum_{p=0}^{i-1} t_{m_p} + \tau_{bit}}{T_j} \right\rceil \right) O_j \tag{2.6}$$

where i is an integer range from 1 to n_m, $hp(m)$ is the set of messages of higher priority than message m, and t_{m_p} is the longest time that message m need to wait on queue from $(p-1)^{th}$ retransmission to p^{th} retransmission before winning arbitrate. The term, t_{m_p}, has already included the transmission time for $(p-1)^{th}$ retransmission.

The difference of two ceiling function correspond to the number of times that message j requires to transmit within the time slot, t_{m_i}. Finally, we get the maximum queuing time:

$$t_{m_i} = \begin{cases} B_m + EF + \sum_{\forall j \in hp(m)} \left\lceil \frac{t_{m_i} + \tau_{bit}}{T_j} \right\rceil O_j & \text{if } i = 0 \\ C_m + EF + \sum_{\forall j \in hp(m)} \left(\left\lceil \frac{\sum_{p=0}^{i} t_{m_p} + \tau_{bit}}{T_j} \right\rceil - \left\lceil \frac{\sum_{p=0}^{i-1} t_{m_p} + \tau_{bit}}{T_j} \right\rceil \right) O_j & \text{otherwise.} \end{cases} \tag{2.7}$$

This model not only considers the effect of higher priority messages' retransmissions on lower priority's response time; but also allows us to arbitrarily assign different number of retransmission on different messages and calculate the worst case response time. This enables us to determine under what circumstances the system will break down. This represents a new and more realistic model which has not been considered previously in the CAN literature.

Illustrative Example

This example consists of a set of 3 periodic messages listed in Table 2.1. For simplicity, we will assume all messages require 1 retransmission before a successful transmission. It will be further assumed the time that an error frame will occupy the bus is equal to one unit of time, and there is a corrupted low priority message on the bus before message 1 release time.

Priority (Id)	C_m	Period	Deadline	R_m
1	2	12	12	9
2	3	24	24	21
3	3	45	45	45

Table 2.1: Message Set

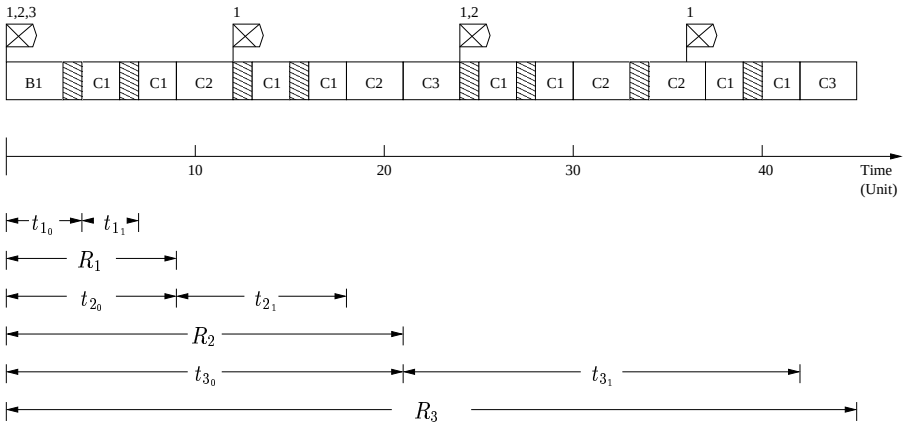


Figure 2.7: Bus Occupation

A graphical interpretation of Eqn.2.5 and Eqn.2.7 is shown in Fig. 2.7 where "pattern box" and "flag" correspond to error frame occupancy and the message release time, respectively. For example, message 1 need to wait for the corrupted message finish transmission before it can start transmitting its content; as a result, t_{1_0} equals to the sum of B_1 and EF. For message 2, it has to wait for message 1's successful transmission before it can proceed its first transmission. However, the first transmission is corrupted and an error frame is received. Since message 1 require to transmit again, message 2 has to wait for message 1's successful transmission before it can proceed its first retransmission. For message 3, it need to wait for all higher priority messages ($m = 1, 2$) finished their transmission and retransmission before it can start transmitting its content. As a result, it will take longer time for it to win the arbitrate as compare to other messages. By using Eqn. 2.5 and Eqn.2.7, we calculate the response time as follow:

m	1	2	3
t_{m_0}	4	9	21
t_{m_1}	3	9	21
R_m	9	21	45

Table 2.2: Analytical results

These results are consistent with Fig. 2.7.

2.2.3 Numerical Example

We are going to consider the same example as in [24]. It is an experimental embedded CAN based application, provided by PSA (Peugeot-Citröen Automobiles Company), that was implemented in a prototype car. Six devices exchange messages: the engine controller, the wheel angle sensor, the AGB (Automatic Gear Box), the ABS (Anti-Blocking System), device y[5] and the body work gateway. The traffic consists of a set of 12 periodic messages (e.g. speed and torque from the engine controller) listed in Table 2.3. The transmission rate of the CAN bus is 250kbps. It will be further assumed that the deadline of each frame is equal to its period. The Data Length Code (DLC) denotes the number of bytes of each frame. Eqn. 2.5 and Eqn. 2.7 enable us to compute the worst case response time for a given number of retransmission as a result of error transmission.

Priority (m)	Transmitter node	DLC	Period	Max. Retry	R_m
1	engine controller	8	10ms	2	2.396ms
2	wheel angle sensor	3	14ms	3	4.032ms
3	engine controller	3	20ms	1	4.804ms
4	AGB	2	15ms	1	5.496ms
5	ABS	5	20ms	4	7.964ms
6	ABS	5	40ms	1	8.896ms
7	ABS	4	15ms	1	9.748ms
8	bodywork gateway	5	50ms	6	18.224ms
9	device y	4	20ms	1	19.076ms
10	engine controller	7	100ms	1	20.088ms
11	AGB	5	50ms	7	34.812ms
12	ABS	1	100ms	2	36.056ms

Table 2.3: PSA message set

In [24], they cannot calculate the worst case response time for messages when

[5]The name of this device cannot be communicated because of confidentiality

corrupted messages lead to retransmission. However, by using Eqn. 2.5 and Eqn. 2.7, we can calculate the worst case response time for messages when the maximum number of retries is known. Thus, our model will able to provide a more realistic timing behavior, as long as we have some idea on how frequent a message will be corrupted.

2.3 Summary

CAN is a deterministic protocol optimized for short messages. The message priority is specified in the arbitration field. Higher priority messages always gain access to the bus during arbitration. Therefore, the transmission delay for higher priority messages can be guaranteed. In addition, it requires less wiring as compared to point to point communication systems. Also, with its excellent error recovery and detection mechanism. CAN protocol has a Hamming distance [16] equal to six. That means it can detect up to 5 errors and correct up to 2 errors (see Appendix A). On average, one undetected error will occur in every 1000 years [4]. Hence, it is highly reliable and its response time is less random. Furthermore, CAN has also been the choice of manufacturers of medical apparatus, textile machines, special-purpose machinery and elevator controls [46, 29].

The major disadvantage of CAN compared with the other networks is the slow data rate (500kbps maximum). Thus the throughput is limited compared with other control networks. This limitation is not due to technological issues, but rather to its physical set-up. The CAN protocol requires the different stations to be synchronized within a bit time; in fact, at any given time all the stations must sense the same physical level on the bus in order the arbitration mechanism to operate correctly. The means that in CAN the transmission rate-bus length product is upper-bounded and his bound depends on the propagation speed of the electrical signals. CAN is

also not suitable for transmission of messages of large data sizes, although it does support fragmentation of data that is more than 8 bytes.

Chapter 3

Process Field Bus

In 1987, the German Federal Minister for Research and Technology requested the collaboration project, 'Field Bus'. It is the digital replacement of the analog 4-20 mA interface which is widely used in industrial process control. Thirteen companies and five institutes worked together to develop an open field bus system under the name PROFIBUS (Process Field Bus) based on the ISO/OSI reference model. The official goal of the project was to quickly propagate PROFIBUS as the field-bus standard. The bus access is based on a hybrid method where masters use a token-passing procedure to grant the bus access and a master-slave procedure to communicate with slave stations. It is well suited for high speed applications using long messages. PROFIBUS can link up to 124 stations with a maximum of 244 bytes input and output data for each slave. It offers high-speed communication rate up to 12Mbits/sec that allows real-time control. PROFIBUS encompasses several Industrial Bus Protocol Specifications, including PROFIBUS-PA, PROFIBUS-FMS, PROFInet and PROFIBUS-DP. PROFIBUS-PA is a full-function fieldbus that is generally used for process level instrumentation. It communicates at 31.25Kbps and has a maximum distance of 1900 meters per segment. PROFIBUS-FMS is a control bus generally used

for communications between DCS (Detector Control Systems) and PLC (Programming Logic Controller) systems, while PROFInet is a protocol being developed to allow PROFIBUS communications across Ethernet Networks. PROFIBUS-DP[43] is a device level bus that supports both analog and discrete signals. PROFIBUS-DP has widespread usage for such items as remote I/O systems, motor control centers, and variable speed drives. PROFIBUS-DP communicates at speeds from 9.6 Kbps to 12 Mbps over distances from 100 to 1,200 meters. For control purposes, we will only analyze the timing behavior of PROFIBUS-DP.

In section 3.1, we describe the PROFIBUS protocol and its way of handling transmission errors. Section 3.2 is devoted to calculating the worst case response time under noise critical environment with an industrial example. Finally, in section 3.3, we will discuss the advantages and disadvantages of using PROFIBUS.

3.1 Protocol Description

This section will discuss some key features in PROFIBUS protocol. It is important to understand how PROFIBUS protocol works because this will enable us to formulate correctly its timing behavior that is crucial to control application. Later in this section, we will discuss PROFIBUS's error handling and recovery process. It will not only affect the timing behavior but also affect the decision making on choosing this network for a particular application.

3.1.1 Fundamental elements in PROFIBUS

PROFIBUS contains master and slave stations. Master devices (active stations) determine the data communication on the bus. A master can send message without an external request when it holds the bus access rights (the token). Slave devices

(passive station) are peripherals such as I/O devices, valves, drives and measuring transducers. They do not have bus access rights and they can only acknowledge received messages or send messages to the master when are requested.

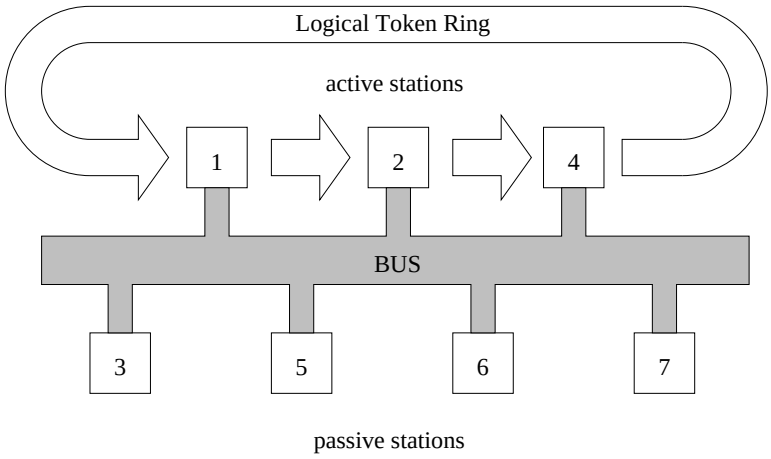


Figure 3.1: PROFIBUS medium access method

As mentioned at the beginning of this chapter, PROFIBUS's protocol is a simplified version of the timed token protocol. A token is passed between masters in ascending order of addresses. This token passing procedure ensures that the bus access right (the token) is assigned to each master within a precisely defined timeframe. The token message, a special telegram for passing the token from one master to the next master, must be passed around the logical token ring once to all masters within a (configurable) maximum token rotation time. In PROFIBUS the token passing procedure is only used for communication between active stations (masters). In the case of PROFIBUS mono-master networks (PROFIBUS-DP,2000), the station just passes the token to itself. The advantage of preserving the same token-passing procedure is that they allow for an unambiguous scheduling of different classes of traffic (high-priority and cyclic/acyclic low-priority), preserving all the properties found in multi-master PROFIBUS networks.

3.1.2 Message Cycle

An important PROFIBUS concept is the Message Cycle, which comprises the Action Frame sent by the initiator (always a master) and the associated Acknowledge or Response Frame sent by the responder. Once the action frame has been transmitted, the initiator waits for response during a Slot Time (T_{SL}). If a response is not received within T_{SL}, initiator will retry up to a number of maximum retry limit. PROFIBUS is capable of distinguishing between high-priority, cyclic low-priority (execution of the requests contained in the poll list), and acyclic low-priority messages. PROFIBUS provides service to poll a list of sensors and actuators, by means of a pre-defined sequence of requests. The processing of all the Poll List entries is said to be a Poll cycle. The Poll Cycle duration depends on the length of each message cycle, the number of message cycles processed at each token arrival and the token rotation time. Hence, a Poll Cycle may last for several token-holding periods. If the Poll Cycle is completed within one token holding period, then the next Poll Cycle may only start at the next receipt of token. Otherwise, the Poll List is processed in segments, without inserting acyclic low-priority message cycles.

3.1.3 Message Dispatching

The amount of time that a master can hold the bus is determined by target rotation time (T_{TR}), real rotation time (T_{RR}), and token holding time (T_{TH}). These variables can be related by the following equation:

$$T_{TH} = T_{TR} - T_{RR} \tag{3.1}$$

The target rotation time, T_{TR}, must be defined in a PROFIBUS network. The value of this parameter is common to all masters. After receiving the token, the measurement

of the token rotation time begins. This measurement expires at the next token arrival and results in the real token rotation time, T_{RR}. When a station receives the token, the token holding time, T_{TH}, timer is given the value corresponding to the difference, if positive, between T_{TR} and T_{RR}. PROFIBUS defines two categories of messages: high priority and low priority. These two categories of messages use two independent outgoing queues. If at the arrival, the token is late, that is, the real token rotation time (T_{RR}) is greater than the target rotation time (T_{TR}), the master station may execute, at most, one high priority message cycle. Otherwise, the master station may execute high priority message cycles while $T_{TH} > 0$. T_{TH} is always tested at the beginning of the message cycle execution. This means that once a message cycle is started it is always completed, including any required retries, even if T_{TH} expires during the execution. The low priority message cycles are executed if there are no high priority messages pending, and while $T_{TH} > 0$.

Example

Let us consider a mono-master Profibus system with two sensors and an actuator in the poll list. In addition to the high priority messages, there is a low priority messages which contain surveillance camera images. The timing description for high and low messages are in Table 3.1. We will assume T_{TR}, T_{RR}^{l-1}, and token latency equal to 10 ms, 4 ms, and 1 ms, respectively.

Name	Symbol	Priority	Trans. Time	Period
Sensor 1	h_1	High	4 ms	20 ms
Sensor 2	h_2	High	4 ms	20 ms
Actuator	h_3	High	3 ms	40 ms
Camera	l_1	Low	5 ms	50 ms

Table 3.1: Timing description

In Fig. 3.2, the black box correspond to token latency (i.e the time that require

to pass a token) and the gray box correspond to the time which T_{TH} is exceeded due to the completion of a message cycle.

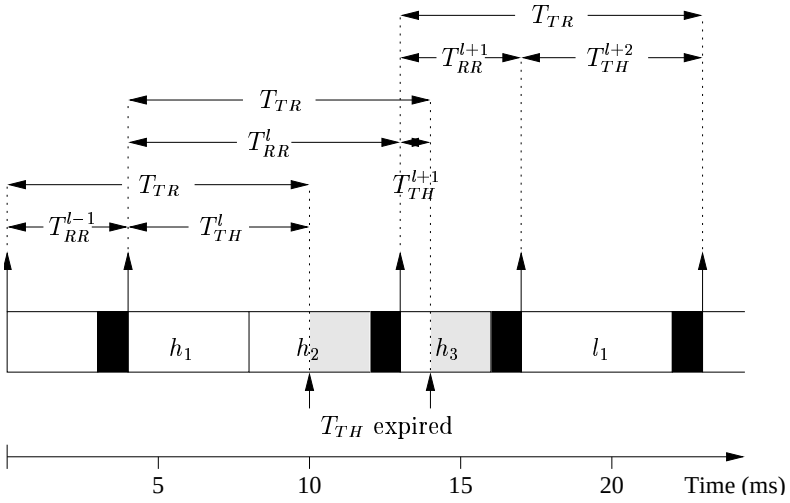


Figure 3.2: Message Cycle

At 4ms, master receives token with token holding time equals to $T_{TR} - T_{RR}^{l-1} = 6ms$. Sensor 1 finished transmission at 8ms. Sensor 2 begin transmission at 8ms. Even though T_{TH}^{l} timer expires at 10ms, sensor 2 must finish transmission before master can pass the token. At 13ms, master receives token with token holding time equals to $T_{TR} - T_{RR}^{l} = 1ms$. Actuator begin transmitting signal and finished at 16ms. Since T_{TH}^{l-1} expires, no more messages can be transmitted. At 17ms, master receives token with token holding time equals to $T_{TR} - T_{RR}^{l-1} = 6ms$. Since $T_{TH} > 0$ and no high priority message pending, camera signal begin transmission.

As demonstrated in the above example, T_{RR} is a variable that depends on the number of messages being sent in the present token cycle. Since T_{TH} depends on T_{RR}, this values will be changing at all time.

3.1.4 PROFIBUS Frame

Data transfer services require standardized frame formats. Together with the most efficient encoding, highly reliable transmission must be ensured. For this purpose the frames are given redundant information which inevitably increases the protocol overhead, thus reducing the net transmission rate of the protocol.

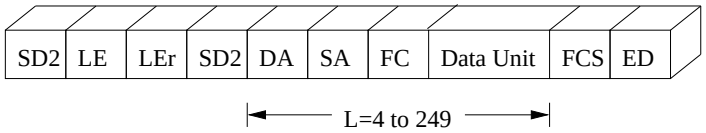


Figure 3.3: Format with variable information field length

A frame always starts with a Start Delimiter (SD) containing code of the frame format. After the SD there are Destination Address (DA) and Source Address (SA) which identify address of receiver and transmitter, respectively. They are followed by the Frame Control (FC), by which the receiver recognizes the frame type (acknowledge frame or response frame). FC contains the frame priority and control information that, for instance, avoids frame loss of the frame. Finally, there is a Frame Check Sequence (FCS) that serves for data security has a 1 byte length, followed by the End Delimiter (ED) which has 1 byte length. The called station acknowledges frame reception by transmitting a frame or a one-character acknowledgment. Fig.3.3 [1] shows an example of maximum frame length that consists of 255 bytes, thus allowing 246 information bytes. However, in the case when masters are passing token to each other, the message frame will not carry any data information. As a result, the token frame will have a length of 3 character.

Unlike CAN, Profibus uses an asynchronous transmission technique, which means that the clock frequencies of transmitter and receiver are not synchronized. The transmission is character-oriented. Therefore, each time the receiver recognizes a

start bit, it must resynchronize its sampling with the sender. Each frame contains a number of characters. In each character, it consists of 1 start bit, 1 stop bit, 1 parity bit and 8 bit of data. Even parity means that the data bits contain an even number of 'ones'. In this case the parity bit is set to zero. In the case that the number of ones is odd, the parity bit is set to one.

3.1.5 Error Detection and Handling

In order to ensure error-free transmission, PROFIBUS provides security with Hamming distance (Hd=4) [8]. In the case of Hd=4, a one-bit error in a character can be detected and corrected, a three-bit error can be detected but not corrected. This is achieved through compliance with the international standard IEC 870-5-1 [7], through special telegram start and end delimiters, slip-free synchronization, a parity bit and a check byte. If an uncorrected error is detected, then PROFIBUS will react according to the availability of services in the data security layer. For example, PROFIBUS-DP data security layer has a service called SRD, Send and Request Data with Reply, that allows a user to transfer data to a single remote station and simultaneously request data from another remote station. If an error occurred, the data transfer will be repeated.

3.2 Timing Analysis

The PROFIBUS-DP profile is aimed at time-critical communications distributed input/output devices. As a result, there is no application layer being developed in this profile, while others include this application layer, that enhance speed and efficiency in the network. The mono-master structure, a single master device cyclically polling many distributed slaves, is often preferred due to its higher baud rate (up

to 12Mbps) [43]. Hence, we will only analyze the timing behavior of mono-master PROFIBUS-DP network. In the next section, we will summarize the results in the literature [40, 41, 22]. After that, we will proposed an improved timing analysis for PROFIBUS-DP.

3.2.1 Literature Review

As the popularity of PROFIBUS in control industry increased, more people realized the significance of understanding the timing behavior of PROFIBUS. Early results on response time analysis was published in various papers [40, 41]. Their analysis are intended for multi-master systems and if applied to the mono-master system, it would lead to very pessimistic results because the authors consider always the worst-case token rotation time. Moreover, none of these works consider the evaluation of response time guarantees for the cyclic poll PROFIBUS messages. In the most recent publication [22], the authors carried out an analysis of timing behavior for mono-master PROFIBUS-DP network. In this section, we will go over some definitions and timing analysis in [22].

Definition 1. (Overrun Window)

We define an overrun window as the time window during which T_{TH} is exceeded due to the completion of a message cycle.

Definition 2. (Late Token)

A token is defined as being late if, at its arrival, the real token rotation T_{RR} is greater than the target token rotation time T_{TR}.

A late token arrival implies that at most one high-priority message can be processed by the related master station. It should be noted that an overrun in a given token arrival, does not usually imply a late token on the next arrival.

Let us denote $A(l)$ as the token arrival instant for the l^{th} token visit. At the time instant $A(l)$, T_{TH}^{l} timer is assigned with the value $T_{TR} - T_{RR}^{l-1}$. Therefore, there will be a late token arrival only if $T_{RR}^{l-1} > T_{TR}$. Note that the real token rotation time is measured between token arrivals.

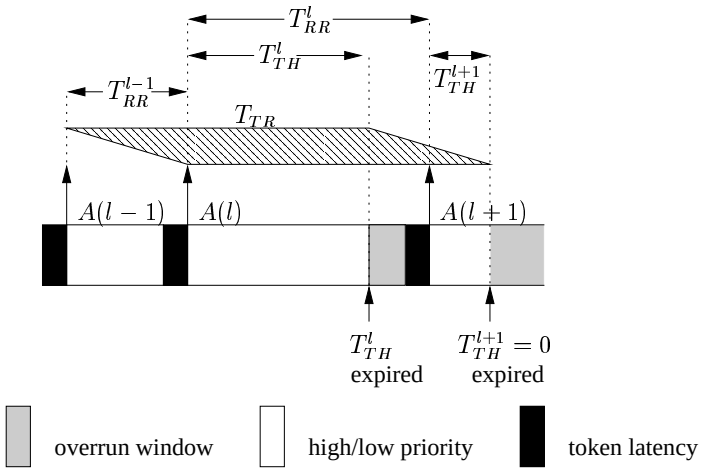


Figure 3.4: Message Cycle and Token Reception

As depicted in Fig. 3.4, it is clear that the token is neither late in the l^{th} token visit nor in the $(l+1)^{th}$ visit, after the one where an overrun has occurred. However, in some particular conditions the token can be late.

Theorem 1. *(Theorem 1 [22])*

In a mono-master Profibus system, if in the l^{th} token visit an overrun occurs, then there will be a late token arrival if and only if the overrun window is greater than the value of T_{RR}^{l-1}.

In [18], the author introduced the concept of critical instant as being the time instant at which a request for a given task has the longest response time, that is, the longest time interval till the end of the response for that request.

Definition 3. (PROFIBUS Critical Instant)

Considering that requests for all high-priority, cyclic and acyclic low-priority message streams are simultaneously placed on the respective outgoing queues. We define a PROFIBUS critical instant as the time instant at which a request for a given message stream has the longest response time.

In a PROFIBUS network, we must consider that, due to FCFS (First Come First Serve) behavior of the outgoing queues, a given message request can be delayed by requests from all the other message streams. Therefore, a critical instant will occur when, for a given priority every message stream simultaneously issue a message request. Moreover, due to non pre-emptive context of messages processing, high-priority request may suffer some additional delay before starting being processed.

Definition 4. (Initial Blocking)

We define the initial blocking as the delay that the first request made at the critical instant may suffer until starting to be processed.

Definition 5. (Critical Load)

We define the critical load for a given priority class, as the time interval between a critical instant and the time instant when the last request (made at the critical instant) for that priority class has been completely processed.

As far as the evaluation of the worst-case response time is concerned, two factors must be taken into account: the initial blocking and the high-priority messages stream made at the critical instant results from the simultaneous occurrence of:

- The longest initial blocking, that is, the first high-priority request suffers the longest possible delay before being processed;
- the longest high-priority critical load, that is, it takes the maximum number of

token visits to process all high-priority requests.

According to Theorem 2 and 3 in [22], the simultaneous occurrence of both conditions leads to the worst-case response time for the last message request to be processed. Since all the possible requests are issued at the critical instant and due to the non pre-emptive context of Profibus, a master may need several token visits to process all high-priority messages, before processing any low-priority request. Thus, there will be a well defined pattern when processing all those requests as depicted in Fig. 3.5.

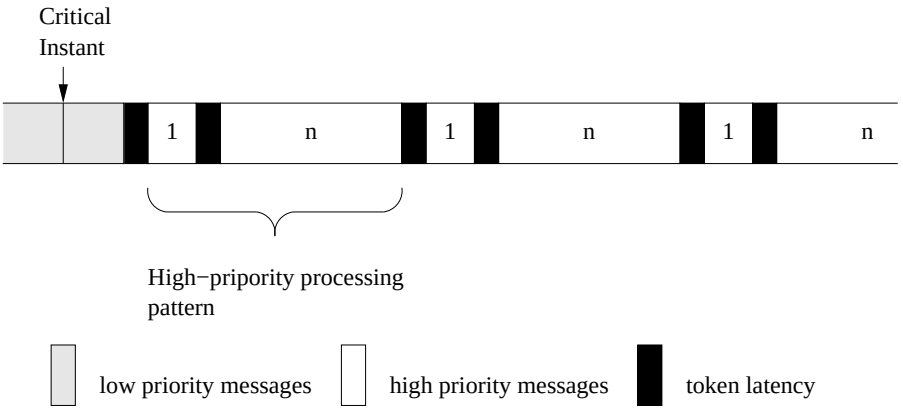


Figure 3.5: The 1 – n processing pattern

This processing pattern is characterized by a late token arrival, where just one high-priority message is processed, followed by an early token arrival, where n high-priority messages are processed.

Theorem 2. *(Theorem 4 [22])*

The occurrence of a Profibus critical instant induces a 1 – n processing pattern for high-priority messages.

If only one high-priority message is processed in the $(l-1)^{th}$ token visit, then the token holding timer is assigned with $T_{TH} = T_{TR} - Ch_{max} - \tau$ in the l^{th} token visit where Ch_{max} is the transmission time for the largest high priority message and

τ is the token passing time. There are $\left\lfloor \frac{T_{TR}-Ch_{max}-\tau}{Ch_{max}} \right\rfloor$ messages processed during this period and one more message will be processed in overrun. Therefore, $n = \left\lfloor \frac{T_{TR}-Ch_{max}-\tau}{Ch_{max}} \right\rfloor + 1 = \left\lfloor \frac{T_{TR}-\tau}{Ch_{max}} \right\rfloor$ high-priority messages will be processed when only one high-priority messages is processed in the last token visit and $\lfloor\ \rfloor$ is the floor function[1].

The worst-case response time for high-priority messages can be computed taking into account the following four components:

1. the initial blocking ($B = Cl_{max} + \tau$) where Cl_{max} and τ are correspond to the longest low priority transmission time and token passing time, respectively;

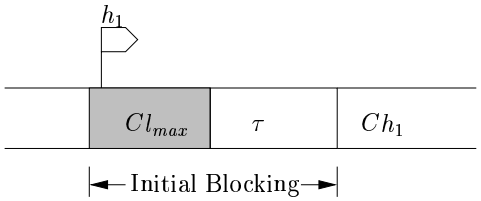


Figure 3.6: Initial Blocking

2. the amount of time require to process n messages with overrun is equal to $T_{TH} + Ch_{max} + \tau = T_{TR}$ and the amount of time requires to process 1 message is $Ch_{max} + \tau$. Therefore, the amount of time require to process $n+1$ messages is equal to $T_{TR} + Ch_{max} + \tau$ and the number of "1–n" processing rounds require to process n_h high priority messages is equal to $\lfloor \frac{n_h}{n+1} \rfloor$;

3. a component Ψ_h which is related to the finishing of the "1 – n" processing pattern[22]. At the end of the last complete cycle of n messages, there are three possible cases:

   (a) there are no more pending requests, and thus the computation of the response time ends before releasing the token in the previous token cycles;

[1] $\lfloor a \rfloor = b$ where b is the biggest integer smaller than or equal to a

(b) there is just one pending request, and thus the time needed to process the pending requests is exactly Ch_{max};

(c) there are more than one pending request, and thus one more token cycle is needed to process the pending requests.

Hence, the worst-case response time for PROFIBUS high priority messages is:

$$R_h = B + \left\lfloor \frac{n_h}{n+1} \right\rfloor \times (T_{TR} + Ch_{max} + \tau) + \Psi_h$$

where:

$$\Psi_h = \begin{cases} -\tau, & \text{if } n_h = \left\lfloor \frac{n_h}{n+1} \right\rfloor (n+1) \\ Ch_{max}, & \text{if } n_h = \left\lfloor \frac{n_h}{n+1} \right\rfloor (n+1) + 1 \\ \left(n_h - \left\lfloor \frac{n_h}{n+1} \right\rfloor (n+1)\right) Ch_{max} + \tau, & \text{otherwise.} \end{cases} \quad (3.2)$$

and

$$n = \left\lfloor \frac{T_{TR} - Ch_{max} - \tau}{Ch_{max}} \right\rfloor + 1 = \left\lfloor \frac{T_{TR} - \tau}{Ch_{max}} \right\rfloor$$

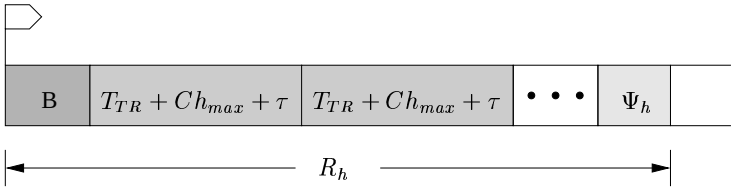


Figure 3.7: Worst case response time

The first case in Ψ_h corresponds to no more message require to be processed; as a result, we don't need to pass a token in the last "1–n" processing round. The second case corresponds to one more message require to be processed. In the last case, there are $\left(n_h - \left\lfloor \frac{n_h}{n+1} \right\rfloor (n+1)\right)$ messages left. In the first token visit, there is only one message processed. Therefore, the rest of the messages require to be processed in the next token visit.

Numerical Example

We are going to consider the same example as in [22]. It is a mono-master PROFIBUS network in an industrial environment, where a decentralized computer controlled system integrates video message streams, computer-generated audio messages and control-related message streams. The supported application controls an assembly production line where parts must be assembled and checked. The control-related message streams, which interconnect sensors and actuators to controllers, are mapped on the high-priority message streams and each message will contain 20 bytes of information. Table 3.2 summarizes the characterization of the high-priority message streams considered in this example, where set x represents a set of message streams with the same periodicity.

	Set 1	Set 2	Set 3	Set 4	Total
Messages per set	3	5	7	5	20
Th_i	20ms	25ms	50ms	60ms	

Table 3.2: Summary of high-priority message streams

We assume the deadline for each message is equal to its period, Th_i, and we will calculate the worst case response time for sending 20 messages. From this worst case response time analysis, we can conclude if messages will or will not meet their deadline. In order to meet realistic requirements, a value for the Slot Time (T_{SL}) and for the target token rotation time (T_{TR}) has been fixed to $100\mu s$ and 8ms, respectively. Moreover, a 1.5Mbps data transfer rate is considered in this example. Therefore, the bit period is equal to $0.667\mu s$ and the computation for the token-passing latency, yields: $\tau = 3 \times (T_{TF} + T_{SL}) = 366\mu s$, where T_{TF} is the token frame length and 3 is the maximum number of retries predefined for the case of the token frame. There are 3 bytes of information in each token frame and T_{TF} is equal to $\frac{3\times 11}{1.5\times 10^6} = 22\mu s$. In addition, the computation of message length for high priority message is as follows:

$Ch_{max} = (9 + 20) \times \frac{11bits}{1.5Mbps} = 213\mu s$, where 9 corresponds to the header and each character contains 11bits.

Concerning message streams to support the video message, we assume the maximum data size for the image is 246 bytes with a period of 50ms. Hence, $Cl_{max} = (9+256) \times \frac{11bits}{1.5Mps} = 1.87ms$ and the maximum blocking is: $B = Cl_{max} + \tau = 2.236ms$. Thus, the number of high-priority message processed in a $1 - n$ processing pattern is: $n + 1 = \left\lfloor \frac{T_{TR} - \tau}{Ch_{max}} \right\rfloor + 1 = 36$. Since the master can process 36 messages per token visit, the master does not require a full round to process all messages, 20. Therefore, the worst-case response time for high priority message streams is

$$\begin{aligned} R_h &= B + \left\lfloor \frac{n_h}{n+1} \right\rfloor (T_{TR} + Ch_{max} + \tau) + \Psi_h \\ &= 2.236ms + \left\lfloor \frac{20}{36} \right\rfloor \times 8.579ms + \left[\left(20 - \left\lfloor \frac{20}{36} \right\rfloor 36 \right) \times 213\mu s + 366\mu s \right] \\ &= 6.855ms \end{aligned}$$

Although their analysis was quiet complete, they did not consider the case when message retransmissions occur. The effect of data retransmission on worst case response time will be discussed in the next section.

3.2.2 Timing Analysis with Message Retransmission

In the previous section, we have presented timing analysis for mono-master PROFIBUS system. However, that analysis ignored the possibility of having retransmissions. In reality, one will expect error transmission will occur from time to time. As a result, timing analysis without considering error transmission is too optimistic. Therefore, we will include message retransmission in our proposed model. We introduce a variable, n_m, as the maximum number of retransmission for a given message.

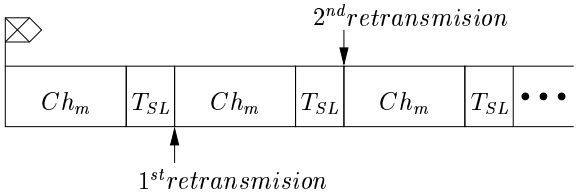


Figure 3.8: Message Retransmission

As we stated in section 3.1.2, a message will be retransmitted if the transmitter does not receive an acknowledgment message from the receiver within T_{SL} after the message is completely dispatched. Furthermore, the master will not release the token unless the current message (include retransmission) is being transmitted successfully. Therefore, the amount of time between message m's 1^{st} transmission and n_m^{th} retransmission is equal to $n_m(Ch_{max} + T_{SL}) + Ch_{max}$. Although we have included the possibility of message retransmission, it does not affect the effect of 1 – n processing pattern. Hence, message retransmission will only affect the number of messages that can be process during this token visit when only one high-priority message is processed in the previous token visit. If only one high-priority message being processed in the $(l-1)^{th}$ token visit, then the token holding timer is assigned with $T_{TH} = T_{TR} - (n_m(Ch_{max} + T_{SL}) + Ch_{max}) - \tau$. Also, in the case when $T_{TH} \gtrsim 0$, one more message will be processed. Therefore, $n = \left\lfloor \frac{T_{TR} - (n_m(Ch_{max}+T_{SL})+Ch_{max}) - \tau}{n_m(Ch_{max}+T_{SL})+Ch_{max}} \right\rfloor + 1 = \left\lfloor \frac{T_{TR}-\tau}{n_m(Ch_{max}+T_{SL})+Ch_{max}} \right\rfloor$ high priority messages will be processed when only one high-priority messages is processed in the last token visit. Hence, the worst-case response

time for PROFIBUS high priority messages with n_m retries is:

$$R_h = B + \left\lfloor \frac{n_h}{n+1} \right\rfloor \times (T_{TR} + (n_m + 1)Ch_{max} + n_m T_{SL} + \tau) + \Psi_h$$

where:

$$\Psi_h = \begin{cases} -\tau, & \text{if } n_h = \left\lfloor \frac{n_h}{n+1} \right\rfloor (n+1) \\ (n_m + 1)Ch_{max} + n_m T_{SL}, & \text{if } n_h = \left\lfloor \frac{n_h}{n+1} \right\rfloor (n+1) + 1 \\ \left(n_h - \left\lfloor \frac{n_h}{n+1} \right\rfloor (n+1)\right)\left((n_m + 1)Ch_{max} + n_m T_{SL}\right) + \tau, & \text{otherwise.} \end{cases}$$

and

$$n = \left\lfloor \frac{T_{TR} - \tau}{(n_m + 1)Ch_{max} + n_m T_{SL}} \right\rfloor \tag{3.3}$$

The three cases in Ψ_h has been described in section 3.2.1. If we assume no retransmission is required then $n_m = 0$ and Eqn. 3.3 will be identical to Eqn. 3.2. Hence it is consistent with the equation derived in [22]. In terms of schedulability, a set of messages maybe schedulable under error free transmission but becomes unschedulable when retransmission is needed. Therefore, message retransmission should always be considered in schedulability test for real-time systems.

Numerical Example

We are going to consider the same example as in [22]. It is a mono-master PROFIBUS network in an industrial environment, where a decentralized computer controlled system integrates video message streams, computer-generated audio messages and control-related message streams. The supported application controls an assembly production line where parts must be assembled and checked. The control-related message streams, which interconnect sensors and actuators to controllers, are mapped

on the high-priority message streams and each message will contain 20 bytes of information. Table 3.3, summarizes the characterization of the high-priority message streams considered in this example, where set x represents a set of message streams with the same periodicity.

	Set 1	Set 2	Set 3	Set 4
n_h=20	3	5	7	5
Th_i	20ms	25ms	50ms	60ms

Table 3.3: Summary of high-priority message streams

In order to meet realistic requirements, a value for the Slot Time (T_{SL}) and for the target token rotation time (T_{TR}) has been fixed to 100μs and 8ms, respectively. Moreover, a 1.5Mbps data transfer rate is considered in this example. Therefore, the bit period is equal to 0.667μs and the computation for the token-passing latency, yields: $\tau = 3 \times (T_{TF} + T_{SL}) = 366\mu s$, where T_{TF} is the token frame length and 3 is the maximum number of retries predefined for the case of the token frame. In addition, the computation of message length for high priority message is as follow: $Ch_{max} = (9 + 20) \times \frac{11bits}{1.5Mbps} = 213\mu s$, where 9 is correspond to header and each character contains 11bits.

Concerning message streams to support the video message, we assume the maximum data size for the image is 246 bytes with period of 50ms. Hence, $Cl_{max} = 1.87ms$ and the maximum blocking is: $B = Cl_{max} + \tau = 2.236ms$. The worst case response time for high priority message with the setting described above are giving in Fig. 3.9.

From the results in Fig. 3.9, we can conclude that each message can retransmit once in order to guarantee all high priority messages meet their deadline. Furthermore, this example has shown message retransmissions will substantially increase the worst case response time for high-priority messages.

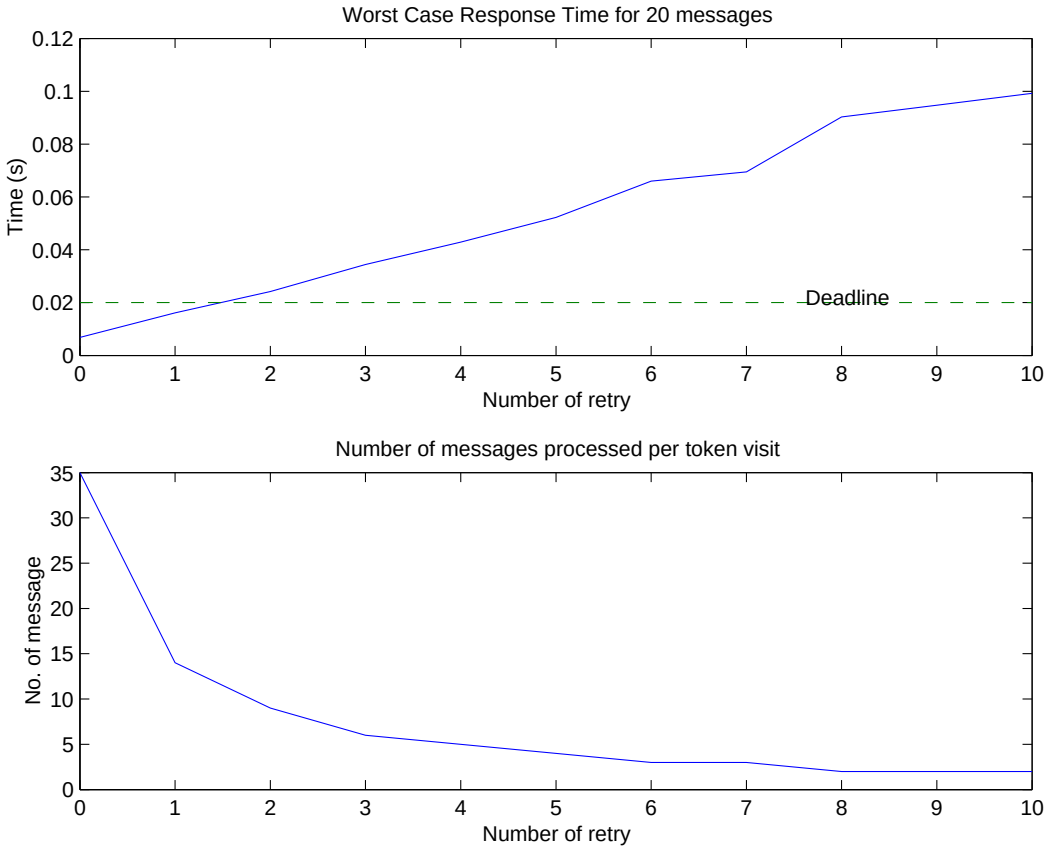


Figure 3.9: Worst case response time and number of message per token visit

3.3 Summary

PROFIBUS is deterministic and provides excellent throughput and efficiency at high network loads [14, 44]. The bus access is based on a hybrid method, where masters use a token-passing procedure to grant the bus access and a master-slave procedure to communicate with slaves. PROFIBUS can handle large amounts of data (up to 244byte) at high speed (up to 12Mbps) and connect up to 124 stations. Furthermore, it is the most widely accepted international networking standard. The major disadvantage of PROFIBUS compare with CAN is its error detection mechanism. It provides security with Hamming distance equals to four (Hd=4) while CAN provides Hd=6.

In this chapter, we have analyzed the timing behavior of mono-master PROFIBUS-DP network. In [40, 41, 22], the authors have developed a timing model for such systems. Since this network is common for industrial application, electromagnetic field in this environment is typically strong. Signal on the bus may be corrupted by electromagnetic interference that cause message retransmission. Therefore, although retransmissions is commonly ignored, we place great importance on including message retransmissions in the model. In our example, we have shown that retransmissions could substantially increase the worst case response time, and our model does provide more realistic results as compared to the model in [40, 41, 22].

Chapter 4

Ethernet

Ethernet is a well-known innovation that emerged from Xerox Corporation's Palo Alto Research Center (PARC) in the 70's, in the early days of computing, . Ethernet transmits a signal between two or more receivers over a shared medium. As a baseband network, Ethernet provides a single channel for communications over the physical medium. Each device has equal access to the medium, and each can use the full available bandwidth; therefore, only one device can transmit at a time. If more than one device transmit signals at the same time, then collision will occur and will be handled by the transport mechanism, Transport Control Protocol (TCP). As the popularity[1] of Ethernet increase, people start to consider applying it to the factory floor. The main advantages are its fast speed, low cost, and people can control a production line from their office. However, this protocol was designed with the needs of Office/Enterprise users in mind. Issues of paramount concern in control networks, such as real-time performance and redundancy were not part of the original designers' concerns. Before applying Ethernet to industrial automation. we have to study its timing behavior.

[1] In [10], the author estimates 95% of the network traffic carried today over wide-area IP networks are using TCP.

In section 4.1, we describe key components in Ethernet. In section 4.2, we present a traditional configuration of Ethernet and demonstrate its disadvantages in control applications. As a result, we present our network configuration that is much more suitable for control applications. In our control network, we have used a LAN switch and full-duplex Ethernet that are described in section 4.3. In section 4.4, we calculate the response time for a set of messages to be successful transmit based on some probabilistic parameters. Finally, in section 4.5, we summarize the key operations in Ethernet control network and its advantages and disadvantages in control application.

4.1 Protocol Description

This section will present the frame structure for Ethernet. Furthermore, we will discuss the connection and collision detection for Ethernet. It is important to understand the basic concept of Ethernet because this will enable us to explain why the traditional configuration, as described in section 4.2, is not suitable for control application.

4.1.1 Frame Structure

An Ethernet frame varies in size from 64 bytes to 1529 bytes. It is made up of the nine fields as shown in Fig. 4.1.

52 bits	8 bits	48 bits	48 bits	16 bits	46−1500 bytes	32 bits
Preamble	SFD	Destination Address	Source Address	Length/ Type	Data or network management information	FCS

Figure 4.1: An Ethernet frame (IEEE 802.3 standard[34])

Preamble

The preamble contains a group of 64 bits that are used to help the hardware synchronize itself with the data on the network. If a few bits of the preamble are lost during transmission, no harm occurs to the message itself.

Start Frame Delimiter (SFD)

SFD marks the end of the preamble and the start of the information-bearing parts of the frame.

Destination address

The destination address contains the physical address of the device that is to receive the frame. The first two bits of this field have special meaning. If the first bit is 0, then the address represents a hardware address of a single device on the network. However, if the first bit is 1, then the address is what is known as a multicast address and the frame is addressed to a group of devices. The second bit indicates where physical device addresses have been set. If the value is 0, then addresses have been set by the hardware manufacturer. When addresses are set by those maintaining the network, the value is 1.

Source address

The source address field contain the hardware address of the device sending the frame.

Length field

The contents of the length field depends on the type of frame. If the frame is carrying data, then the length field indicates how many bytes of meaningful data are present.

However, if the frame is carrying management information, then the length field indicates the type of management information present in the frame.

Data field

The data field carries a minimum of 46 bytes and a maximum of 1500 bytes. If there are less than 46 bytes of data the field will be padded to the minimum length.

Frame Check Sequence (FCS)

The last field (also known as a cyclical redundancy check (CRC) field) contains 32 bits used for error checking. The bits in this field are set by the transmitting device based on the pattern of bits in the data field. The receiving device then regenerates the FCS. If what the receiving device obtains does not match what is in the frame, then some bits were changed during transmission and some type of transmission error has occurred. Unlike CAN and Profibus, Ethernet does not provide the option for error correction. Therefore, if there is any bit error in a frame, then the entire frame will be discarded and request for retransmission will be sent to sender.

4.1.2 Connection Method

TCP is a connection-oriented protocol in which a user and the network set up a logical connection before transfer of data occurs. A connection is described by four factors: source-address, destination-address, source-port, and destination-port. In order to manage a connection, the involved instances need to distinguish the packets and determine which one belongs to which connection, according to the packet's sequence number. Every packet has a 32-bit sequence number that is inserted by the sending host. Sequence number are not unique as 32-bit values can have only 4 billion different states. The sending hosts also increments the sequence numbers by

extra bit(s), depending on the system. Since packets do not stay in channel forever, their sequence number can be used again when they reach the targets. The most interesting point is how those connections have to be established, and that's where the three-way handshake appears. Fig. 4.2 illustrates the major operations between two TCP entities establishing a connection.

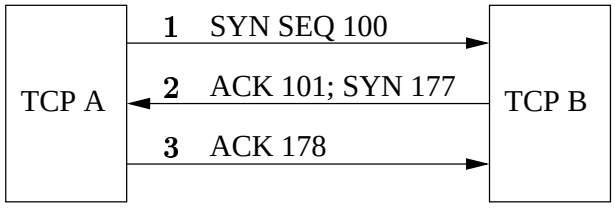


Figure 4.2: TCP operation

Invoking an event, active open, requires TCP A to prepare a segment with the SYN bit set to 1. The segment is sent to TCP B and is depicted in the figure as **1** and coded as SYN SEQ 100. In this example, sequence (SEQ) 100 is used as the initial send sequence (IS) number. The ISS number is to be used for subsequent numbering of user data. Upon receiving the SYN segment number, TCP B returns an acknowledgment with sequence number of 101. It also sends its ISS number 177. This event is labelled as **2**. Upon receipt of this segment, TCP A acknowledges with a segment containing the acknowledgment number 178, depicted as event **3** in the figure. This acknowledgment indicate that the connection has been established. In section 4.1.3, we will discussed the situation when more than one device want to transmit simultaneously.

4.1.3 Detection Method

As stated in the beginning, Ethernet is a baseband network that provides a single channel for communications over the physical medium. Therefore, collisions may occur when more than one device transmit signals at the same time. This

problem is resolved by CSMA/CD (Carrier Sense Multiple Access/ Collision Detection). CSMA/CD means that, before transmitting, each device or station first senses whether there is network traffic. If the network is clear, transmission proceeds immediately; otherwise, the device waits a specified period of time before transmitting. If more than one devices run through the same routine and transmit simultaneously, a collision occurs, and a predetermined "backoff" algorithm governs the rules for retransmission. If a message frame detects a collision for the first time, then the frame will wait for a time duration of $T_{backoff}$ before retransmission. When second collision occur for the same frame, the backoff time will be increased to twice the previous backoff time. This process will continue until the maximum number of retries is reached. For example, if $T_{backoff} = 5ms$, then the waiting time for retransmission, after 4^{th} collision, is $2^{4-1} \times 5ms$. As a consequence of collision's random behavior, the processing time for a message is highly indeterministic.

4.2 Network Configuration

In this section, we will present a traditional network connection and explain the reason why this is not suitable for control application. After that, we will present our network configuration that will reduce response time and improve reliability.

4.2.1 Traditional Configuration

A possible assumption is that the "control network", including the controller, sensors, and actuators, can be attached to the "standard network", such as network printers, personal computers, and internet, while controlled system behavior is not disturbed by non-control network traffic. The suggested configuration is as shown in Fig. 4.3.

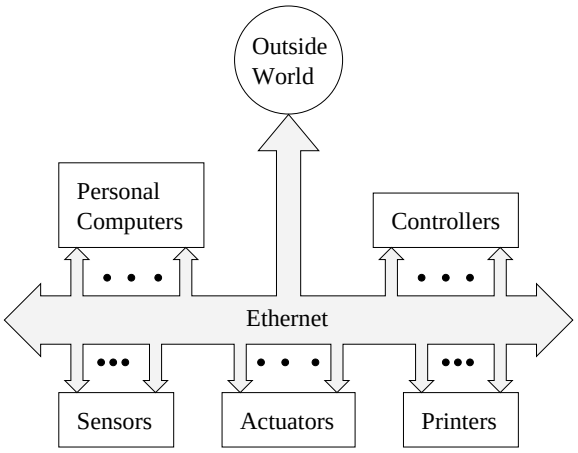


Figure 4.3: Network configuration

In reality, performance of the controlled system is likely to be affected. For example, imagine a company that design and manufacture graphics cards. The designers send large files through the network regularly, and very often the transmission would use up most of the network's capacity, causing slow transmission for the other parties. This is typical in Ethernet networks that a single intensive user can affect the entire network stems. Consequently, this will increase network delay among controller, sensors, and actuators. The basic plan of network is that each information packet sent from a computer is seen by all the other computers on the local network for its address. This scheme is simple, but has performance consequences as the size or level of activity of the network increases. Hence, this network configuration is not suitable for control applications.

4.2.2 Control Network Configuration

In this section, we will present the network model that will be used for timing analysis. In the previous section, we have explained the reason why traditional network configuration is not suitable for control application. The main reason is that a single

intensive user can affect the entire network stems. Consequently, network delay will be increased. Therefore, the most simple and effective solution is to separate the network into two sub-networks, one for the control system and one for the office. A router links the two networks and connects both to the internet, as shown in Fig. 4.4.

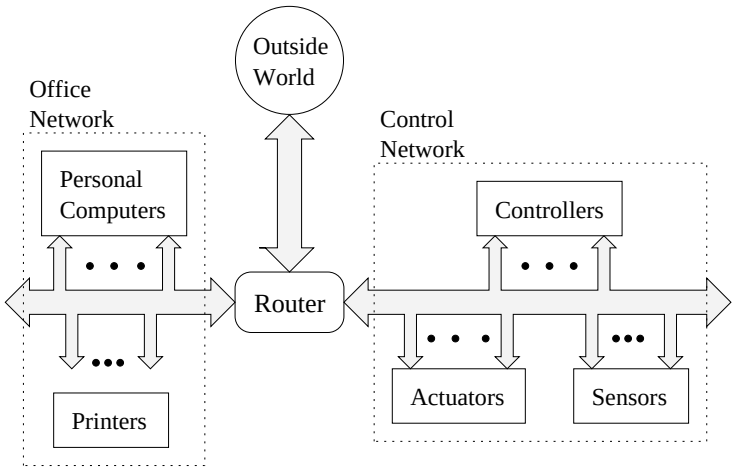


Figure 4.4: Network with router

The router is the only device that sees all message sent by any computer on either of the company's networks. When a designer sends a huge file to another designer, the router looks at the recipient's address and keeps the traffic on the office network. When a designer, on the other hand, probes a sensor for a measurement, the router sees the recipient's address and forwards the message between the two networks. The router ensures that information would not go where it is not needed. This is crucial for keeping large volumes of data from clogging the connections of "innocent bystanders".

We can separate a company's network into two sub-networks, namely, the control network and the office network. However, from a control perspective, we only care about the control network configuration because it is the only part which affects system performance. Let us consider the network configuration as shown in Fig. 4.4. If a sensor is transferring data to controller, then all the other sensors that want to

transfer data to controller have to wait until the bus becomes idle. This behavior is called half-duplex, which means that data could be transmitted in only one direction at a time. Obviously, this is not very efficient. In order to increase the efficiency, Local Area Network (LAN) Switch is introduced to the network.

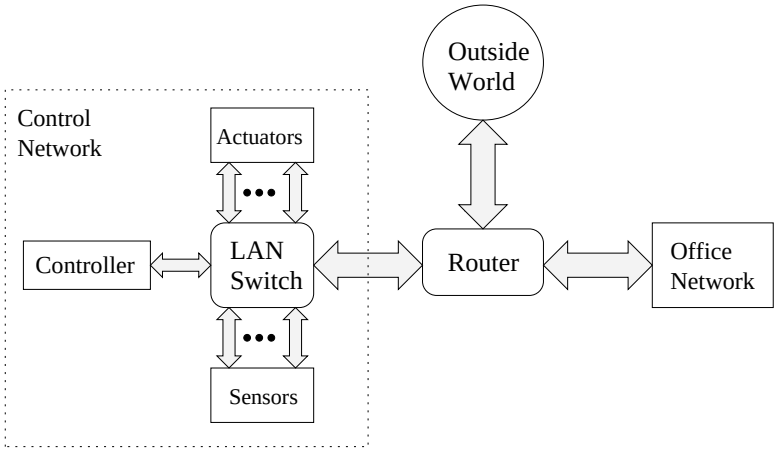


Figure 4.5: Network with switch and router

In a fully switched network (star network), as shown in Fig. 4.5, switches replace all the hubs of an Ethernet network with a dedicated segment for every node. These segments connect to a switch, which supports multiple dedicated segments (sometimes in the hundreds). Since the only devices on each segment are the switch and the node, the switch picks up every transmission before it reaches another node. The switch then forwards the frame over the appropriate segment. Since any segment contains only a single node, the frame only reaches the intended recipient. This allows many conversations to occur simultaneously on a switched network. Another major advantage of switching hubs is that they can link LAN segments that run at different speeds.

All switches contain some high-speed buffer memory in which a frame is stored, however, before being forwarded onto another port or ports of the switching hub.

This mechanism is known as store-and-forward switching. A larger amount of buffer memory allows a switch to handle longer streams of back-to-back frames. This gives the switch improved performance in the presence of bursts of traffic on the LAN. When a switch operates at store-and-forward mode, all frames will be checked for any bit errors that have occurred during frame transmission and reception. This will prevent switches forward faulty messages to the receiver. Although this configuration will prevent signal collision between different nodes, it does not prevent signal collision between incoming and outgoing signals from the same node. This collision will increase transmission time, especially in a network that required rapid data exchange, such as control network.

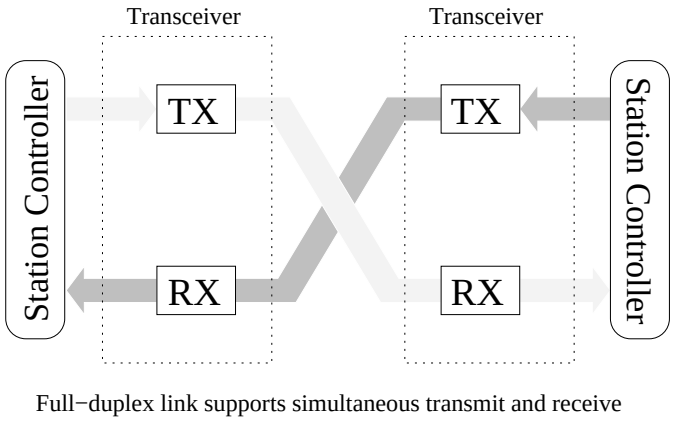


Figure 4.6: Full duplex operation[35]

Fortunately, fully switched networks can employ either twisted-pair or fiber-optic cabling, both of which use separate conductors for sending and receiving data. This is called full-duplex network, as shown in Fig. 4.6. Information can travel from node to switch and from switch to node simultaneously, and the network is collision-free. As a result, the aggregate bandwidth is doubled. Since frame collision will never occur, CSMA/CD is being disabled in full-duplex Ethernet. In the situation where a number of sensors simultaneously send signal to a controller, incoming packets are

saved to the input port temporary memory (buffer) in the switch. The packets in the buffer will be forwarded to the output port controller in first in first out (FIFO) manner.

Throughout this chapter, we will assume the control network is a fully switched network as described in above. This setting is also suggested in [23, 5, 3]. The main two reasons for [23, 5] to choose full-duplex switched network as a control network are: eliminate data collisions and increase the network's efficiency. The switch eliminates the problem of network determinism by providing full bandwidth with storage to a node or group of nodes. In addtion, this setting eliminates all collisions that typically make Ethernet nondeterministic. Also, as stated in [3], today's Ethernet installations (100Mbps, full duplex in a switched network) show latency measured in microseconds – many orders of magnitude better than most factory floor reliability requirements. These articles confirmed that our network setting is a possible solution for Ethernet to be used as a control network.

In this section, we have brifely introduced the full-duplex switched network. However, in some cases where senders transmit signals at a higher rate than a switch can process, this will lead to "buffer overflow". Fortunately, switch has a "PAUSE" mechanism that can resolve this problem and will be described in section 4.3.

4.3 Full-Duplex Mode

After we have described the control network setting, we are ready to take a close look on the timing behavior. As we described in the previous section, there are only two components in each segment of the network and they are LAN switch and a station. However, in the case where multiple senders transmit messages to a single receiver, buffer in the switch will lead to buffer overflow problem. In this section, we will discuss the special operation, PAUSE [32], that is implemented in full-duplex

Ethernet to resolve the problem.

4.3.1 Buffer Overflow

In section 4.2.1, we have mentioned that messages coming from sender must be stored in the switch's buffer and forwarded to receiver. In the case when multiple messages have the same destination address, messages will be stored in a buffer and forwarded to the receiver in FIFO manner. In the situation when there is more messages than the buffer can accommodate, this is called "Buffer Overflow". In the traditional Ethernet configuration (half-duplex mode), all incoming messages will be discarded when the buffer is full. However, in full-duplex mode, media access control (MAC) has defined a special operation to handle buffer overflow.

4.3.2 MAC Control Protocol

The optional MAC Control portion of the 802.3x supplement provides a mechanism for real-time control and manipulation of the frame transmission and reception process in an Ethernet station. In normal Ethernet operation, the media access control (MAC) protocol defines how to go about transmitting and receiving frames. In the optional Ethernet flow control system, the MAC Control protocol provides mechanisms to control when Ethernet frames are sent.

When implemented, the MAC Control system provides a way for the station to receiver a MAC Control frame and act upon it, The operation of the MAC Control system is transparent to the normal media access control functions in a station. MAC Control is not used for a non-real-time function like configuration interfaces, which is handled by network management mechanisms. Instead, MAC Control is designed to allow stations to interact in real time to control the flow of traffic. New functions beyond flow control may be added in the future. In full-duplex mode, MAC issues a

special operation, PAUSE, that is used to resolve buffer overflow problem.

4.3.3 PAUSE Operation

The PAUSE system of flow control on full-duplex link segments is defined in 802.3x and uses MAC Control frames to carry the PAUSE commands. A station that receives a MAC Control frame with this opcode in the first byte of the data field knows that the control frame is being used to implement the PAUSE operation, for the purpose of providing flow control on a full-duplex link segment.

The PAUSE function is specifically designed to prevent switches (or end stations) from unnecessarily discarding frames due to input buffer over flow under short-term transient overload conditions. Consider a device designed to handle the expected steady-state traffic of the network, plus an allowance for a some time variation of that load. The PAUSE function allows such a device to avoid discarding frames even when the short-term load increases above the level anticipated by the design. The device can prevent buffer overflow by sending PAUSE frames to the partner which will cause the partner to stop sending data frames. This gives the first device time to reduce its buffer congestion either by processing frames in the queue that are destined for the device (end station operation).

4.3.4 Overview of PAUSE Operation

The PAUSE operation implements a very simple stop-start form of flow control. A device wishing to temporarily inhibit incoming data sends a PAUSE frame, with a parameter indicating the length of time that the full duplex partner should wait before sending any more data frames. When a station receives a PAUSE frame, it stops sending data frames for the period of time specified as the parameter in the frame. When this timer expires, the station resumes sending data frames where it

left off.

A station issuing a PAUSE may cancel the remainder of the PAUSE period by issuing another PAUSE frame with a parameter of zero time. That is, newly received PAUSE frames override any PAUSE operation currently in progress. Similarly, the issuing station can extend the PAUSE period by issuing another PAUSE frame with a non-zero time parameter before the first PAUSE period has expired. After we have discussed PAUSE operation, we are going to explain when should a station send out or cancel a PAUSE frame.

4.3.5 Buffer Thresholds

A typical input-queued switch will have some amount of buffering available for each port, which holds frames until either the output port or the switching fabric itself is available to accept the frame, as depicted in Fig. 4.7.

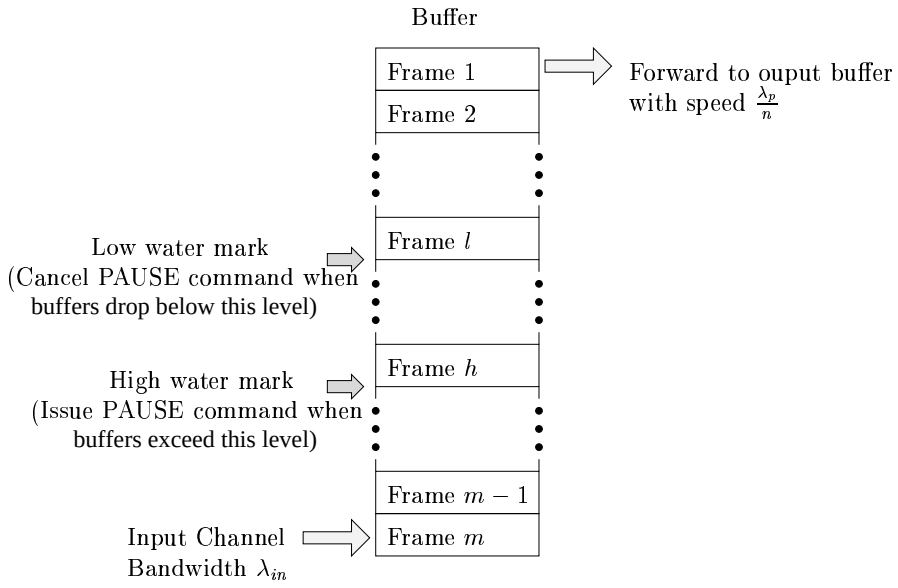


Figure 4.7: Input buffer with PAUSE operation

Depending on the traffic patterns and the total load offered to the switch, frames will experience a delay in this queue while waiting to be unloaded. During this time,

additional frames are being received on the same port, causing the queues to fill further. A reasonable flow control policy would be to send a PAUSE frame (with a non-zero value for the pause time) when the buffer fills up to a predetermined high-water mark, so that the switch can prevent frames from being dropped at the input due to buffer unavailability. while the link partner is throttled in the manner, the switch will be unloading frames from this queue and forwarding them out other ports of the switch. When the buffer empties below a predetermined low-water mark, the flow control can be cancelled (by sending a PAUSE frame with a 0 value for the pause time), and normal operation resumes. In this manner, the switch can be used at maximum capacity without discarding any frames.

As discussed earlier, the high-water mark should be set such that there is still sufficient buffering available above the mark to accommodate the additional traffic that may still be in the pipeline. This is a function of the data rate, media type, and length of the link. Similarly, to ensure that buffer starvation does not occur, there should be sufficient room below the low-water mark that incoming frames can arrive before the queue is completely emptied. Buffer underflow is less of a problem than overflow, since starvation will only cause a minor performance degradation rather than incurring and end-to-end recovery as would be need in the event of a frame discard on overflow. The room required below the low-water mark is also less predictable, since it is a function of how fast he switch can empty the queue. This depends to a great extent on the specific switch architecture, whether there are output queues at all, the speed of the switching fabric between input and output queues, and the data rate of the output port.

4.4 Timing Analysis

After we have described the full-duplex Ethernet, we are ready to take a close look on the timing behavior. For simplicity, we assume every stations transmit the same amount of data at the same time. First, we will calculate the response time for n stations simultaneously transmit to a target station with each station transmitting q packets. The response time analysis is based on the knowledge of output bandwidth (λ_{out}), input bandwidth (λ_{in}), and switch's processing speed (λ_p). The above variables are in frames per second. After that, we will explain the relationship between packet loss and bandwidth.

4.4.1 Assumption

In control systems, sensors and actuators signals are periodically updated by controllers. Therefore, it is naturally to assume every stations begin transmission at the same time. For simplicity, we will assume n stations simultaneously transmit to a target station with each station transmitting q packets. Each input/output port has a buffer that can stored m packets. Furthermore, we assume the knowledge of input bandwidth (λ_{in}), output bandwidth (λ_{out}), and switch's processing speed (λ_p) is available. Every input buffers have a equal probability to be served by the processor at any moment. Hence, we can assume each input buffers are served by the processor at a rate of $\frac{\lambda_p}{n}$. Fig. 4.8 shows the input and output relationship with variables in place.

There are some assumptions on LAN's switch behavior that are described below. The switch monitors its input and output buffers to prevent buffer overflow. For output buffers, if the buffer is full, then the switch will stop transferring frames from input buffers to output buffer. The switch will resume transferring frames from input buffers to output buffers, as soon as the output buffer is no longer full. However,

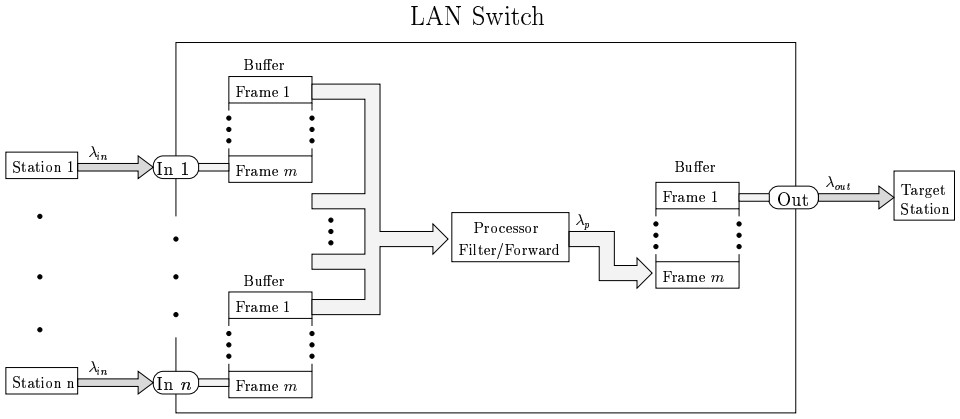


Figure 4.8: LAN Switch (Input/Ouput relationship)

for input buffers, if the buffer reaches high water mark (h), then PAUSE frame will be transmitted to sender and one more message will be received from sender due to PAUSE frame latency. After sender received PAUSE frame, sender cannot start transmission until input buffer reaches low water mark (l) or the predefined pause time expired. This operation has been described in section 4.3.5 and shown in Fig. 4.7. Furthermore, we assume all input buffers have equal chance to be processed by the switch.

4.4.2 Output Buffer Behavior

In the case where switch's processor has a higher frame transfer rate than the output port's rate ($\lambda_p > \lambda_{out}$), the output buffer will eventually get full. It is because the output channel cannot catch up the processor's speed. On the other hand, if the output channel can digest messages faster than processor can handle, then the output buffer will never fill up. As a result, the amount of time to fill up output buffer with size m is given as follow:

$$T_{FO} = \begin{cases} \frac{m}{\lambda_p - \lambda_{out}} & \text{if } \lambda_p > \lambda_{out}, \\ \infty & \text{otherwise.} \end{cases} \tag{4.1}$$

Example

For example, the output port is connected to 100Mbps Ethernet and each frame (packet) has a size of 12204 bits (maximum frame size). Therefore, the output bandwidth is 8194 frames per second. Furthermore, we use Cisco Catalyst 3000 Ethernet switch that can process 14880 frames per second and has a buffer size of 16 frames. Since the processing speed is higher than output sending rate, the output buffer will be filled up, eventually. The amount of time require to filled up the buffer is

$$\begin{aligned} T_{FO} &= \frac{16 frames}{14880 frames/sec - 8194 frames/sec} \\ &= 2.393 ms \end{aligned}$$

By the time that output buffer fill up, the receiver has already received 2.393ms $\times$ 8194 frames/sec =19 frames.

4.4.3 Input Buffer Behavior

After we have analyzed the output buffer behavior, we will study the impact of output buffer on input buffer. In a switch, there is only one single processor that transfer input buffers frames to output buffers.

The input buffer behavior can be divided into four cases, that are:

1. output and input buffers never fill up,

2. output buffer fills up but input buffers never fill up,

3. output buffer never fills up and input buffers fill up,

4. output and input buffers fill up.

Case I

In the case where output buffer never fills up ($\lambda_{out} \geq \lambda_p$) and input buffers never fill up ($\lambda_{in} \leq \frac{\lambda_p}{n}$), the amount of time required for the switch to receive q packets from each sender is $\frac{q}{\lambda_{in}}$.

Example

For example, a network with 10 senders and a switch is connected by 10Mbps Ethernet cable. The switch and a receiver is connected by 1Gbps Ethernet cable. Each frame contains 12204 bits of information plus header. Therefore, the input and output bandwidth is 819 and 81940 frames per second, respectively. Furthermore, we use Cisco Catalyst 3000 Ethernet switch that can process 14880 frames per second and has a buffer size of 16 frames. Since $\lambda_{out} > \lambda_p$ and $\frac{\lambda_p}{10} > \lambda_{in}$, the input and output buffers never fill up. Therefore, the amount of time require for the switch to receive 50 packets from each sender is $\frac{50 frames}{819 frames per sec} = 0.061 sec$

Case II

In the case where the output buffer fills up ($\lambda_{out} < \lambda_p$), the amount of time require to fill up output buffer is $T_{FO} = \frac{m}{\lambda_p - \lambda_{out}}$. After output buffer fills up, output buffer must transmit a frame to the receiver before it can received a frame from input buffers. The total amount of time required for a frame to transmit from input to output buffer is equal to $\frac{1}{\lambda_p} + \frac{1}{\lambda_{out}}$. Hence, from the input buffers point of view, the effective processing rate is given as follow

$$\frac{1}{\lambda_{ep}} = \frac{1}{\lambda_{out}} + \frac{1}{\lambda_p} \tag{4.2}$$

Since this effective processing rate is being shared among all input buffers with equal probability, each input buffer has a processing rate of $\frac{\lambda_{ep}}{n}$. Therefore, we can ensure the input buffers never fill up if the effective processing rate is greater than or equal to input channel rate ($\frac{\lambda_{ep}}{n} \geq \lambda_{in}$). The amount of time required for the switch to receiver q packets from each sender is $\frac{q}{\lambda_{in}}$.

Example

For example, a network with 5 senders and a switch is connected by 10Mbps Ethernet cable. The switch and a receiver is connected by 100Mbps Ethernet cable. Each frame contains 12204 bits of information plus header. Therefore, the input and output bandwidth is 819 and 8194 frames per second, respectively. Furthermore, we use Cisco Catalyst 3000 Ethernet switch that can process 14880 frames per second and has a buffer size of 16 frames. Since $\lambda_{out} < \lambda_p$, the output buffer will fill up. The amount of time require to fill up output buffer is $T_{FO} = \frac{16}{14880-8194} = 2.39ms$. After the output buffer filled up, the effective processing rate becomes $\lambda_{ep} = \frac{8194 \times 14880}{8194+14880} =$ 5284 frames per second. The output buffers never fill up becase $\frac{\lambda_{ep}}{n} > \lambda_{in}$. The amount of time required for the switch to receiver 50 packets from each sender is $\frac{50 frames}{819 frames per second}$=0.061sec.

Case III

In the case where output buffer never fills up, input buffers will still have a chance to get filled up. It is because one single processor is being shared by n different input ports. Therefore, on average, messages transfer from each input to output buffer has

a rate of $\frac{\lambda_p}{n}$. If $\frac{\lambda_p}{n} < \lambda_{in}$, then input buffer will reach high water mark, eventually. As a result, the amount of time require for input buffer to reach high water mark, h, is given as follow:

$$T_h = \frac{h}{\lambda_{in} - \frac{\lambda_p}{n}} \tag{4.3}$$

After sender receives PAUSE frame, sender stops transmitting frames to the switch. At the same time, the processor will transmit frames in input buffers to output buffer. After input buffer level reaches low water mark, l, or PAUSE time, T_{pause}, expired, sender will resume for transmission. The amount of time required for senders to resume transmission is given as follow:

$$T_r = min \left\{ \frac{h + 1 - l}{\frac{\lambda_p}{n}}, T_{pause} \right\} \tag{4.4}$$

Since $\frac{\lambda_p}{n} < \lambda_{in}$, input buffer will reach high water mark again. Therefore, buffer level will fluctuate between lower and higher water mark and the amount of time require for buffer to reach high water mark again is given as follow:

$$T_{\text{r-to-h}} = \frac{\left\lfloor T_r \frac{\lambda_p}{n} \right\rfloor}{\lambda_{in} - \frac{\lambda_p}{n}} \tag{4.5}$$

The numerator corresponds to the number of frames space available in input buffer before it reaches high water mark. The use of a floor function in the above equation is to model that a switch does not accept or store partial frame.

Example

For example, a network with 10 senders and a switch is connected by 100Mbps Ethernet cable. Each frame contains 12004 bits of information plus header. Therefore, the input bandwidth is 8194 frames per second. Furthermore, we use Cisco Catalyst 3000 Ethernet switch that can process 14880 frames per second and has a buffer size of 16 frames. The high and low water marks are set to 14 and 8 frames level. The PAUSE time, T_{pause}, is set to 10ms. Since $\lambda_{in} = 8194 > \frac{\lambda_p}{n} = 1488$, the input buffer will cross the high water mark. The amount of time require for input buffer to reach high water mark is

$$\begin{aligned} T_h &= \frac{14 frames}{8194 frames/sec - \frac{14880 frames/sec}{10}} \\ &= 2.088 ms \end{aligned}$$

After the switch sent PAUSE frame to the sender, the amount of time require for senders to resume transmission is

$$\begin{aligned} T_r &= \min \left\{ \frac{(14 + 1 - 8) frames}{\frac{14880 frames/sec}{10}}, 10ms \right\} \\ &= 4.704 ms \end{aligned}$$

When buffer level reaches low water mark, sender starts transmitting frames, again. The amount of time require for the buffer to reach high water mark is equal to

$$T_{\text{r-to-h}} = \frac{\left\lfloor \frac{4.704ms}{1488frames/sec} \right\rfloor}{8194frames/sec - \frac{14880frames/sec}{10}}$$
$$= 0.895ms$$

Case IV

In the case where the output buffer fills up, the output buffer must transmit a frame to receiver before it can received a frame from input buffers. The total amount of time required for a frame to transmit from input to output buffer is equal to $\frac{1}{\lambda_p} + \frac{1}{\lambda_{out}}$. Hence, from the input buffers point of view, the effective processing rate is given as follow

$$\frac{1}{\lambda_{ep}} = \frac{1}{\lambda_p} + \frac{1}{\lambda_{out}} \tag{4.6}$$

Since this effective processing rate is being shared among all input buffers with equal probability, each input buffer has a processing rate of $\frac{\lambda_{ep}}{n}$. If $\frac{\lambda_{ep}}{n} < \lambda_{in}$, then input buffers will be filled up eventually. At the time when output buffer fills up, input buffer has $\left\lfloor T_{FO} \times \left(\lambda_{in} - \frac{\lambda_p}{n}\right) \right\rfloor$ frames. Therefore, the amount of time require for input buffer to reach high water mark is given as follow:

$$T_h = \begin{cases} \frac{h}{\lambda_{in} - \frac{\lambda_p}{n}} & \text{if } \frac{h}{\lambda_{in} - \frac{\lambda_p}{n}} < T_{FO}, \\ \frac{h - \left\lfloor T_{FO}\left(\lambda_{in} - \frac{\lambda_p}{n}\right) \right\rfloor}{\lambda_{in} - \frac{\lambda_{ep}}{n}} + T_{FO} & \text{otherwise.} \end{cases} \tag{4.7}$$

The first term in the Eqn. 4.7 corresponds to the case where input buffer filled up faster than output buffer. The second term corresponds to the case where output

buffer filled up faster than input buffer. Since the input buffer level will fluctuate between low and high water mark, the amount of time require to reach low water mark will be different at different time. Therefore, we will denote T_r^i as the i^{th} time for senders to resume transmission, and it is given as follow:

$$T_r^i = \begin{cases} min\left\{\frac{h+1-l}{\frac{\lambda_{ep}}{n}}, T_{pause}\right\} & \text{if } T_h + \sum_{k=0}^{i-1}(T_r^k + T_{\text{r-to-h}}^k) \geq T_{FO} \\ min\left\{T_{\text{pcd}}^i, T_{pause}\right\} & \text{if } T_h + \sum_{k=0}^{i-1}(T_r^k + T_{\text{r-to-h}}^k) + min\left\{\frac{h+1-l}{\frac{\lambda_p}{n}}, T_{pause}\right\} > T_{FO} \\ min\left\{\frac{h+1-l}{\frac{\lambda_p}{n}}, T_{pause}\right\} & \text{otherwise.} \end{cases} \tag{4.8}$$

where $T_{\text{r-to-h}}^i$ is correspond to the time require to reach high water mark again for the i^{th} time and T_r^0 equals to 0. The summation term, $\sum_{k=0}^{i-1}(T_r^k + T_{\text{r-to-h}}^k)$, in the augmanted statments is corresponds to the time between the 1^{st} and i^{th} time reaches high water mark. The first case correspond to the case where output buffer has already filled up before PAUSE frame sent out. In the second case, T_{pcd}^i is given by

$$T_{\text{pcd}}^i = \frac{r}{\frac{\lambda_p}{n}} + \frac{h+1-r-l}{\frac{\lambda_{ep}}{n}} \tag{4.9}$$

where r is the smallest integer satisfies this equation, $T_h + \sum_{k=1}^{i-1}(T_r^k + T_{\text{r-to-h}}^k) + \frac{r+1}{\frac{\lambda_p}{n}} > T_{FO}$. This is correspond to the case where output buffer became full after it received r frames. This case describes the time require to reach low water mark while the output buffer becomes full in the mean time that changes the processing rate of the switch. The last case correspond to output buffer will not full during this high to low water mark process.

In order to calculate the amount of time require to reach high water mark, we

need to know the input buffers level after senders resume transmission. We denote, n_{in}^{i}, as the input buffers' level (after i^{th} pause period) which can be calculated by the following formula:

$$n_{in}^{i} = \begin{cases} max\left\{l, h+1-\left\lfloor T_{pause}\frac{\lambda_{ep}}{n}\right\rfloor\right\} & \text{if } T_h + \sum_{r=0}^{i-1}(T_r^r + T_{\text{r-to-h}}^r) \geq T_{FO} \\ n_{pcd}^{i} & \text{if } T_h + \sum_{r=0}^{i-1}(T_r^r + T_{\text{r-to-h}}^r) + min\left\{\frac{h+1-l}{\frac{\lambda_p}{n}}, T_{pause}\right\} > T_{FO} \\ max\left\{l, h+1-\left\lfloor T_{pause}\frac{\lambda_p}{n}\right\rfloor\right\} & \text{otherwise.} \end{cases} \tag{4.10}$$

The first term in Eqn. 4.10 corresponds to the input buffers' level when output buffer is full. In the second term, n_{pcd}^{i} is given by

$$n_{pcd}^{i} = \begin{cases} h+1-\left\lfloor T_{pause}\frac{\lambda_p}{n}\right\rfloor & \text{if } T_{pause} \leq \frac{r}{\frac{l_p}{n}} \\ h+1-r-\left\lfloor\left(T_{pause}-\frac{r}{\frac{\lambda_p}{n}}\right)\frac{\lambda_{ep}}{n}\right\rfloor & \text{if } \frac{r}{\frac{l_p}{n}} < T_{pause} \leq \frac{r}{\frac{l_p}{n}} + \frac{h+1-r-l}{\frac{l_{ep}}{n}} \\ l & \text{otherwise} \end{cases} \tag{4.11}$$

where r can be solved by Eqn. 4.9. The first case corresponds to the pause time expired before output buffer filled up. The second case corresponds to the pause time expired after output buffer filled up. The last case corresponds to the case where input buffers return to low-water mark before pause time expired. The second case in Eqn. 4.10 corresponds to the input buffers' level when output buffer fills up during the pause period. The last term corresponds to the input buffers' level when output buffer is not full.

Similarly, the resume transmission to high water mark process can be divided into three cases. The first case corresponds to the case where output buffer has already

filled up. The second case, T^i_{pci}, corresponds to amount of time require to reach high water mark while the processing rate changes in the mean time. The last case corresponds to output buffer will not be full during this resume transmission to high water mark process. Therefore, the amount of time required for buffer to reach high water mark after senders resume transmissions is given as follow:

$$T^i_{\text{r-to-h}} = \begin{cases} \frac{h-n^i_{in}}{\lambda_{in}-\frac{\lambda_{ep}}{n}} & \text{if } T_h + \sum_{r=1}^{i}(T^r_r + T^{r-1}_{\text{r-to-h}}) \geq T_{FO} \\ T^i_{\text{pci}} & \text{if } T_h + \sum_{r=1}^{i}(T^r_r + T^{r-1}_{\text{r-to-h}}) + \frac{h-n^i_{in}}{\lambda_{in}-\frac{\lambda_p}{n}} > T_{FO} \\ \frac{h-n^i_{in}}{\lambda_{in}-\frac{\lambda_p}{n}} & \text{otherwise.} \end{cases} \tag{4.12}$$

and $T^0_{\text{r-to-h}}$ equals to 0.

$$T^i_{\text{pci}} = \frac{r}{\lambda_{in}-\frac{\lambda_p}{n}} + \frac{h-r-n^i_{in}}{\lambda_{in}-\frac{\lambda_{ep}}{n}} \tag{4.13}$$

where r is the smallest integer satisfies this equation,$T_h + \sum_{k=1}^{i}(T^k_r + T^{k-1}_{\text{r-to-h}}) + \frac{r+1}{\frac{\lambda_p}{n}} > T_{FO}$. This is correspond to the case where output buffer became full after it received r frames.

Example

For a switch with 10 senders and 1 receiver connect to a Cisco Catalyst 3000 switch, that can process 14880 frames per second and has a buffer size of 16 frames, through 100Mbps Ethernet cable. The high and low water marks are set to 14 and 8 frames level. The PAUSE time, T_{pause}, is set to 100ms. It will take 2.393ms to fill up output buffer. Therefore, after output buffer filled up, the effective process rate is equal to $\frac{8194\times14880}{8194+14880} = 5284$ frame per second. By using Eqn. 4.7, we can calculate

the amount of time for input buffer to reach high water mark and it is equal to 2.088ms. Since $T_h + \frac{h+1-l}{\frac{\lambda_p}{n}} = 6.792ms > 2.393ms = T_{FO}$, it is case two in Eqn. 4.8. Therefore, the amount of time required to reach low water mark is equal to $min\{13.2ms, 100ms\} = 13.2ms$. According to Eqn. 4.12, it will take 0.783ms to reach high water mark, again.

4.4.4 Effective Bandwidth

As we described in section 4.2.2, there are only two components in each segment of a fully switch network (star network) and they are a LAN switch and a station. Each station uses a separate channel for sending and receiving data; as a result, packets loss should never occur. Although the network setting provide a reliable and error free transmission, bit error will occur from time to time. It is because mechanical and electrical components generate magnetic field that interfere signal on the bus. In addition, any error in a frame will lead to retransmission. This will substantially increase the response time and make the effective bandwidth smaller. Bandwidth is defined by the amount of information transmitted in a period of time. However, this definition is not really useful to our analysis. It is because error frames will not be processed and stored in buffer.

Definition 6. (Effective Bandwidth) *It is defined by the amount information, except error frame, transmitted in a period of time.*

When sending a frame in full-duplex mode, the station ignores carrier sense and does not defer to traffic being received on the channel. However, the station still waits for an interframe gap period between frame transmissions as Ethernet interfaces are designed to expect an interframe gap between each frame. Providing the interframe gap ensures that the interfaces at each end of the link can keep up with the full frame

rate of the link. The interframe gap is a 96 bit time delay[35] provided between frame transmissions. Also, if we know there will be p error frames out of n frames, then the total transmission time for n frames plus p retransmissions is given as follow:

$$T_{transmission} = \frac{(n+p)(FS+96)}{C} \tag{4.14}$$

where C is the bandwidth and FS is the frame size in bits. Since there are only $n \times FS$ bits of useful information, the effective bandwidth is

$$\text{Effective Bandwidth} = \frac{n \times FS}{T_{transmission}} \tag{4.15}$$

Example

For example, a 100Mbps Ethernet network that transmits frames with size of 12204 bits (maximum frame size) with an error frame rate of 1/1000 (measured from Ethernet cable). The total transmission time for 1000 frames plus 1 retransmission is equal to 0.1231 second. Therefore, the effective bandwidth is equal to $\frac{1000 frames}{0.1231 sec} = 99.1 Mbps$.

4.4.5 Response Time

After we analyzed the timing behavior within LAN switch, we have enough information to calculate the amount of time required for n senders transmit q packets through a switch to a receiver, simultaneously. In section 4.4.3, we have calculated the amount of time for PAUSE operation to be activated and deactivated. During PAUSE time, senders are not allowed to transmit any messages to the switch. Therefore, this PAUSE time substantially increase the amount of time require for senders

to dispatch their messages completely. After reaching the high water mark for the first time, the input buffer level will fluctuate between high and low water mark. As a result, there will be a "transmit-pause" pattern. Once we calculated the amount of time for each transmit round, we can determine the number of messages being transmitted during each round. Hence we can calculate the number of rounds required for n stations to transmit q packets.

The response time analysis can be divided into five parts:

1. amount of time, T_1, required for the first bit of data to travel from sender to switch (propagation delay),
2. amount of time, T_2, required for switch to receive all frames,
3. amount of time, T_3, required to clear all frames in input buffer,
4. amount of time, T_4, required to clear all frames in output buffer.,
5. amount of time, T_5, required for the last bit of data to travel from switch to receiver (propagation delay).

The amount of time, T_1, required for the first bit of data travel from senders to switch is depends on the physical distance. The propagational speed for electrical signal travel through wire is around $2 \times 10^8 m/s$. T_5 can be calculated in the similar manner. Since the input and output buffers behavior have a direct impact on response time analysis, we divide the analysis into four cases, that are:

1. output and input buffers never fill up,
2. output buffer fills up but input buffers never fill up,
3. output buffer never fills up and input buffers fill up,
4. output and input buffers fill up.

Case I

In this case, output buffer never fills up ($\lambda_{out} \geq \lambda_p$), and input buffers never fill up ($\lambda_{in} \leq \frac{\lambda_p}{n}$). Since the input buffers never fill up, PAUSE operation will never be activated. Therefore, the amount of time, T_2, required for switch to receive q packets in each input buffers is equal to $\frac{q}{\lambda_{in}}$. After the last bit of data is received by input buffers, there will be one frame left in each input buffers. The amount of time, T_3, required to clear the last frames from input buffers is equal to $\frac{1}{\frac{\lambda_p}{n}}$. Similarly, there will be one frame left in output buffer. The amount of time, T_4, required to clear the last frame from the output buffer is equal to $\frac{1}{\lambda_{out}}$. Therefore, the total response time is equal to $T_1 + T_2 + T_3 + T_4 + T_5$.

Case II

In the case where the output buffer fills up ($\lambda_{out} < \lambda_p$), the amount of time require to fill up output buffer is $T_{FO} = \frac{m}{\lambda_p - \lambda_{out}}$. After output buffer fills up, output buffer must transmit a frame to the receiver before it can received a frame from input buffers.

$$\frac{1}{\lambda_{ep}} = \frac{1}{\lambda_{out}} + \frac{1}{\lambda_p} \tag{4.16}$$

Since this effective processing rate is being shared among all input buffers with equal probability, each input buffer has a processing rate of $\frac{\lambda_{ep}}{n}$. Therefore, we can ensure the input buffers never fill up if the effective processing rate is greater than or equal to input channel rate ($\frac{\lambda_{ep}}{n} \geq \lambda_{in}$). The amount of time required, T_2, for the switch to receiver q packets from each sender is $\frac{q}{\lambda_{in}}$. Since input buffers never fill up, only one frame needed to be processed in each buffer. The amount of time require to process the last frame is given as follow:

$$T_3 = \begin{cases} \frac{1}{\frac{\lambda_p}{n}} & \text{if } T_2 < T_{FO}, \\ \frac{1}{\frac{\lambda_{ep}}{n}} & \text{otherwise.} \end{cases} \tag{4.17}$$

After $T_2 + T_3$, no more messages will be transmitted to the output buffer. For the output buffer, the buffer is full if $T_2 + T_3 \geq T_{FO}$. However, if $T_2 + T_3 < T_{FO}$, then the number of frames in buffer is equal to $\lfloor (T_2 + T_3)(\lambda_p - \lambda_{out}) \rfloor$. Hence, the amount of time required to clear the output buffer is equal to:

$$T_4 = \begin{cases} \frac{\lfloor (T_2+T_3)(\lambda_p-\lambda_{out}) \rfloor}{\lambda_{out}} & \text{if } T_2 + T_3 < T_{FO}, \\ \frac{m}{\lambda_{out}} & \text{otherwise.} \end{cases} \tag{4.18}$$

Case III

In the case where output buffer never fills up, input buffers will still have a chance to get filled up. It is because one single processor is being shared by n different input ports. Therefore, on average, messages transfer from each input to output buffer has a rate of $\frac{\lambda_p}{n}$. In section 4.4.3, we have calculated the amount of time required to reach high water mark (T_h), the amount of time that senders have to wait before resume transmission after the i^{th} visit of high water mark (T_r^i), and the time required to reach high water mark again after i^{th} resume transmission ($T^i_{\text{r-to-h}}$). T_r^i corresponds to the time where sender cannot send messages. Therefore, the number of rounds, r, required for n senders to transmit q frames from each sender to switch can be calculated by the following equation:

$$\lfloor T_h \lambda_{in} \rfloor + \sum_{i=0}^{k} \left\lfloor T^i_{\text{r-to-h}} \lambda_{in} \right\rfloor \geq q \tag{4.19}$$

where k is the smallest integer satisfies Eqn. 4.19 and λ_{in} is the input channel effective bandwidth. Hence, the amount of time required for a switch to receiver q frames from n senders can be calculated by the following equation:

$$T_2 = \begin{cases} T_h + \sum_{i=0}^{k-1} \left(T^i_{\text{r-to-h}} + T^i_r\right) + T^k_r + \frac{Res}{\lambda_{in}} & = \text{if } r > 0, \\ \frac{q}{\lambda_{in}} & \text{otherwise.} \end{cases} \tag{4.20}$$

where k is defined by Eqn. 4.19 and $Res = q - \lfloor T_h \lambda_{in} \rfloor - \sum_{i=0}^{k-1} \lfloor T^i_{\text{r-to-h}} \lambda_{in} \rfloor$ as the number of messages remain after $k-1$ rounds. The term, T_h, in the first case corresponds to the amount to time require for input buffers to reach high water mark for the 1^{st} time. The second term, $\sum_{i=0}^{k-1} \left(T^i_{\text{r-to-h}} + T^i_l\right)$, corresponds to the time between 1^{st} and k^{th} time reaches high water mark.

In order to calculate the amount of time, T_3, required to clear all frames in input buffer, we need to determine the number of messages in the buffer when the last frame received by the switch and it is given as follow:

$$n_3 = \begin{cases} \left\lfloor \frac{q}{\lambda_{in}} \left(\lambda_{in} - \frac{\lambda_p}{n}\right) \right\rfloor & \text{if } \lfloor T_h \lambda_{in} \rfloor \geq q, \\ h + 1 - \left\lfloor T_r \frac{\lambda_p}{n} \right\rfloor + \left\lfloor \frac{Res}{\lambda_{in}} \left(\lambda_{in} - \frac{\lambda_p}{n}\right) \right\rfloor & \text{otherwise.} \end{cases} \tag{4.21}$$

The first case corresponds to the case where q frames have been received by the switch before input buffers reach high water mark. The term, $h + 1 - \left\lfloor T_r \frac{\lambda_p}{n} \right\rfloor$, in the second

case corresponds to the input buffers' level after $(k-1)^{th}$ pause period. The last floor term in the second case corresponds to the number of frames increase in the input buffers. Therefore, the amount of time, T_3, required to clear all the frames in input buffers is equal to $\frac{n_3}{\frac{\lambda_p}{n}}$. Since the output buffer never fills up, only one frame is left in the output buffer after input buffers have been cleared. The amount of time, T_4, required to clear the last frame in output buffer is equal to $\frac{1}{\lambda_{out}}$.

Case IV

In the case where the output buffer fills up, the output buffer must transmit a frame to receiver before it can received a frame from input buffers. The total amount of time required for a frame to transmit from input to output buffer is equal to $\frac{1}{\lambda_p}+\frac{1}{\lambda_{out}}$. Hence, from the input buffers point of view, the effective processing rate is given as follow

$$\frac{1}{\lambda_{ep}} = \frac{1}{\lambda_p} + \frac{1}{\lambda_{out}} \tag{4.22}$$

Since this effective processing rate is being shared among all input buffers with equal probability, each input buffer has a processing rate of $\frac{\lambda_{ep}}{n}$. If $\frac{\lambda_{ep}}{n} < \lambda_{in}$, then input buffers will be filled up eventually. The number of rounds required for n senders to transmit q frames from each sender to switch can be calculated by the following equation:

$$\lfloor T_h \lambda_{in} \rfloor + \sum_{i=0}^{k} \left\lfloor T^i_{\text{r-to-h}} \lambda_{in} \right\rfloor \geq q \tag{4.23}$$

where k is the smallest integer satisfies Eqn. 4.23 and λ_{in} is the input channel effective

bandwidth. The amount of time required for a switch to receive q frames from n senders can be calculated by the following equation:

$$T_2 = \begin{cases} T_h + \sum_{i=0}^{k-1} \left(T^i_{\text{r-to-h}} + T^i_r\right) + T^k_r + \frac{Res}{\lambda_{in}} & = \text{if } r > 0, \\ \frac{q}{\lambda_{in}} & \text{otherwise.} \end{cases} \tag{4.24}$$

where k is defined by Eqn. 4.23 and $Res = q - \lfloor T_h \lambda_{in} \rfloor - \sum_{i=0}^{k-1} \lfloor T^i_{\text{r-to-h}} \lambda_{in} \rfloor$ as the number of messages remain after $k-1$ rounds. In order to calculate the amount of time, T_3, required to clear all frames in input buffer, we need to determine the number of messages in the buffer when the last frame received by the switch and it is given as follow:

$$n_3 = \begin{cases} n^k_{in} + \left\lfloor \frac{Res}{\lambda_{in}} \left(\lambda_{in} - \frac{\lambda_p}{n}\right) \right\rfloor & \text{if } k \neq 0 \text{ and } T_2 < T_{FO} \\ \left\lfloor \frac{q}{\lambda_{in}} \left(\lambda_{in} - \frac{\lambda_p}{n}\right) \right\rfloor & \text{if } k = 0 \text{ and } T_2 < T_{FO} \\ \left\lfloor \left(\lambda_{in} - \frac{\lambda_p}{n}\right) T_{FO} \right\rfloor + \left\lfloor \left(\frac{q}{\lambda_{in}} - T_{FO}\right) \left(\lambda_{in} - \frac{\lambda_{ep}}{n}\right) \right\rfloor & \text{if } k = 0 \text{ and } T_2 \geq T_{FO} \\ n^k_{in} + \left\lfloor \frac{Res}{\lambda_{in}} \left(\lambda_{in} - \frac{\lambda_{ep}}{n}\right) \right\rfloor & \text{if } k \neq 0 \text{ and } T_2 - \frac{Res}{\lambda_{in}} > T_{FO} \\ n^k_{in} + \left\lfloor \frac{Res - n_{ep}}{\lambda_{in}} \left(\lambda_{in} - \frac{\lambda p}{n}\right) \right\rfloor + n_{ep} & \text{otherwise.} \end{cases} \tag{4.25}$$

where $n_{ep} = \left\lfloor (T_2 - T_{FO}) \left(\lambda_i n - \frac{\lambda_{ep}}{n}\right) \right\rfloor$ is the increment of input buffers' level after the output buffer filled up and k can be determined by Eqn. 4.23. The first case corresponds to the case where output buffer does not fill up and PAUSE operation has been activated in the input buffers. The second case corresponds to the case where PAUSE operation does not activate in the input buffers and output buffers

does not fill up. The third case corresponds to the case where PAUSE operation does not activate and output buffers filled up. The fourth case corresponds to the case where output buffer filled up in the past $k-1$ rounds and PAUSE operation has been activated in the input buffers. The last case corresponds to the case where output buffer fills up during the k^{th} round and PAUSE operation has been activated. Therefore, the amount of time that input buffer required to clear n_3 frames is given as follow:

$$T_3 = \begin{cases} \frac{n_3}{\frac{\lambda_p}{n}} & \text{if } T_2 + \frac{n_3}{\frac{\lambda_p}{n}} \leq T_{FO} \\ \frac{k}{\frac{\lambda_p}{n}} + \frac{n_3 - k}{\frac{\lambda_{ep}}{n}} & \text{otherwise.} \end{cases} \tag{4.26}$$

where k is the smallest integer satisfies the inequality $T_2 + \frac{k+1}{\lambda_p} > T_{FO}$.

After input buffers transmit all the frames in their buffer to the output buffer, the number of frames remain in the output buffer is equal to $min\left\{\lfloor (T_2+T_3)(\lambda_p - \lambda_{out})\rfloor, m\right\}$. Therefore, the amount of time required to clear all the frames remain in the output buffer is equal to

$$T_4 = \frac{min\left\{\lfloor (T_2+T_3)(\lambda_p - \lambda_{out})\rfloor, m\right\}}{\lambda_{out}} \tag{4.27}$$

4.4.6 Example

Consider a control network (fully switched) with 10 sensors and 1 controller. Input and output ports are connected to 100Mbps full-duplex Ethernet cable. Each sensor is required to send 50 frames. Each frame contain 1500 bytes of data plus 204bts header. The frame error rate is 0.001 in each cable. The switch that we assume to be used is Cisco Catalyst 300 Ethernet switch that can process 14880 frames per second

and has a buffer size of 16 frames for each port. We have set the high water mark and low water mark for input buffers at 14 and 9 frames, respectively. The effective bandwidth is 99.1Mbps. The total response time for 10 sensors to transfer 50 frames per sensor to a controller is 0.0941s, as shown in Table 4.1.

Time used	T_1	T_2	T_3	T_4	T_5	Notes
$5\mu s$	✓					1^{st} frame arrived to switch
$2.1ms$		✓				17 frames arrived to switch's input buffers and reached high water mark
$13.3ms$		✓				PAUSE period
$0.79ms$		✓				6 frames arrived in each input buffers
$13.3ms$		✓				PAUSE period
$0.79ms$		✓				6 frames arrived in each input buffers
$13.3ms$		✓				PAUSE period
$0.79ms$		✓				6 frames arrived in each input buffers
$13.3ms$		✓				PAUSE period
$0.79ms$		✓				6 frames arrived in each input buffers
$13.3ms$		✓				PAUSE period
$0.79ms$		✓				6 frames arrived in each input buffers
$13.3ms$		✓				PAUSE period
$0.37ms$		✓				3 frames arrived in each input buffers
$0.019s$			✓			Clear the remaining 10 frames in each input buffers
$2ms$				✓		Clear the remaining 16 frames in output buffer
$5\mu s$					✓	Last bit arrived to receiver
$0.1074\ s$	✓	✓	✓	✓	✓	Total time used

Table 4.1: Response Time

4.5 Summary

In this chapter, we have introduced Ethernet as a possible network for control application. In particular, we have provided a detailed network configuration which is much more reliable and efficient for control application. We have separated a network into office and control network by a router. Furthermore, in the control network, we

placed a LAN switch in between any two devices (e.g. actuators, sensors and controllers). This kind of network is called fully switched network and we take advantage of using full-duplex Ethernet. Full-duplex Ethernet has two physical media for reception and transmission that ensure signal collision will never occur. Furthermore, we have provided a timing analysis which accounts for the PAUSE operation.

The main advantages for this control network are high speed transmission, large data size as compare to CAN and PROFIBUS, and remote access to control components. This control setting provides the most reliable connection in the Ethernet world as described in [3, 23]. However, it is less reliable than CAN and PROFIBUS because error correction mechanism is not implemented in Ethernet protocol. Furthermore, the cost for this network is probably two to three times more than a traditional Ethernet network.

Chapter 5

Controller Design

In Chapter 2, 3, and 4, we have discussed that different networks will have different levels and types of delay. In general, the characteristics of time delays can be constant, bounded, or even random, depending on the network protocols and the chosen hardware. It is well-known in control systems that time delays can degrade a system's performance and even cause system instability. Therefore, a more complicated design approach is necessary to include time delay. Through this chapter, we will only discuss discrete-time controller design because we assume controller will be implemented in a digital processor.

This chapter is divided into three sections. In Section 5.1, we will introduce the networked control system model, major assumption and variables that will be used through the chapter. In Section 5.3, we will discuss controller design for system (e.g. CAN and Profibus) with constant delay. This controller is evolved from the famous Smith predictor [19, 33]. In Section 5.5, we will discuss controller design for system (e.g. Ethernet) with random delay. In particular, we will only consider the case when the delay is bounded and governed by a Markov process. Also, the normal sense of stability and detectability are not directly applicable to this stochastic system. Therefore, we will present the notion of mean square stability for stochastic systems.

At the end of each sections, an example will be used to illustrate controller design for constant or random delay.

5.1 System Model

Consider the block diagram of a networked control system with a single controller, but with multiple sensors and actuators as shown in Fig. 5.1. There are n states (x), m inputs (u), and r outputs (y) in the plant model, and q states (z), r inputs (w), and m outputs (v) in the controller dynamics model. We use s_r and a_m to represent the sensor-to-controller and controller-to-actuator delays, respectively. The variables w_i and u_j are the delayed y_i and v_j signals, $i = 1, ..., r$ and $j = 1, ..., m$, respectively.

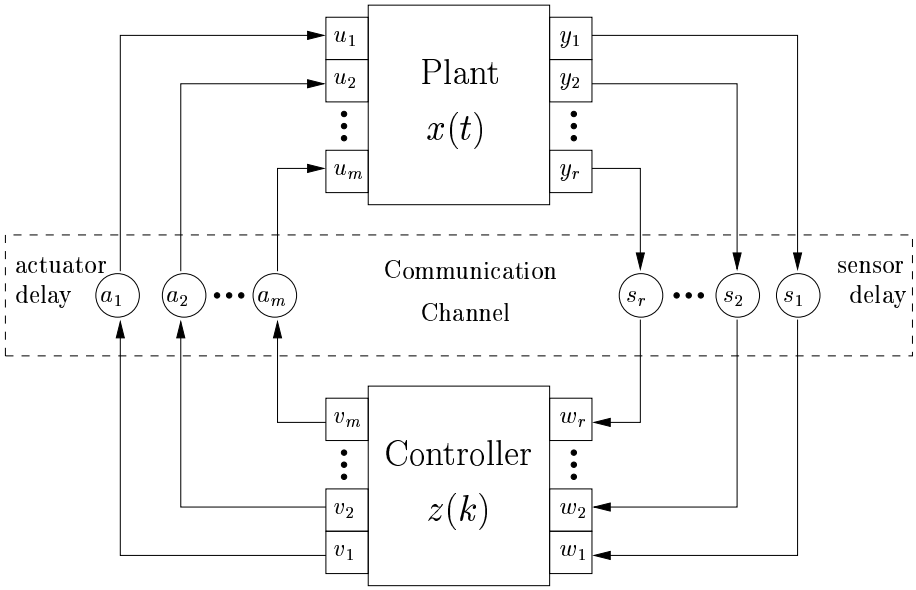


Figure 5.1: Block diagrams of a networked control system

In Fig. 5.1, the continuous time state-space model of the linear time-invariant

plant can be described by the following standard form:

$$\begin{aligned} \dot{x}(t) &= A_c x(t) + B_c u(t) \\ y(t) &= C_c x(t) \end{aligned} \tag{5.1}$$

where $x(t) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^m$, $y(t) \in \mathbb{R}^r$ and the constant matrices A_c, B_c, and C_c. Since the controller is implemented on a digital computer, the controller is designed in discrete time with a sampling time T and the state-space model of the dynamic controller can be expressed as follows:

$$\begin{aligned} z(k+1) &= Fz(k) + Gw(k) \\ v(k) &= Hz(k) + Jw(k) \end{aligned} \tag{5.2}$$

where $z(k) = z(kT) \in \mathbb{R}^q$, $w(k) = w(kT) \in \mathbb{R}^r$, $v(k) = v(kT) \in \mathbb{R}^m$ and the constant matrices F, G, H, and J.

In practical applications of networked control systems, a sensor signal will be sampled and stored in a memory slot embedded in a I/O card attached with the sensor. Afterwards, the I/O card will wait for the availability of the communication channel to send out information from its memory slot to the controller's memory slot. Then, the controller will calculate the corresponding gain for each actuators based on the information that are available in the memory slot at that sampling instant. For simplicity, we assume the sensors and actuators have the same sampling time, T, as the controller and the upper bound of delay is known. This timing behavior is illustrated in Fig. 5.2.

For an amplifier with gain of two, the closed loop control sequence is shown in Fig. 5.2. Also, in the figure, we assume the delay is shorter than one sampling period and control actions initiate at time $(k-1)T$. Fig. 5.2a corresponds to analog signal measured by sensor. The sampled sensor's signal, with period of T, is shown in Fig. 5.2b. This sampled signal will be transmitted through a communication channel to

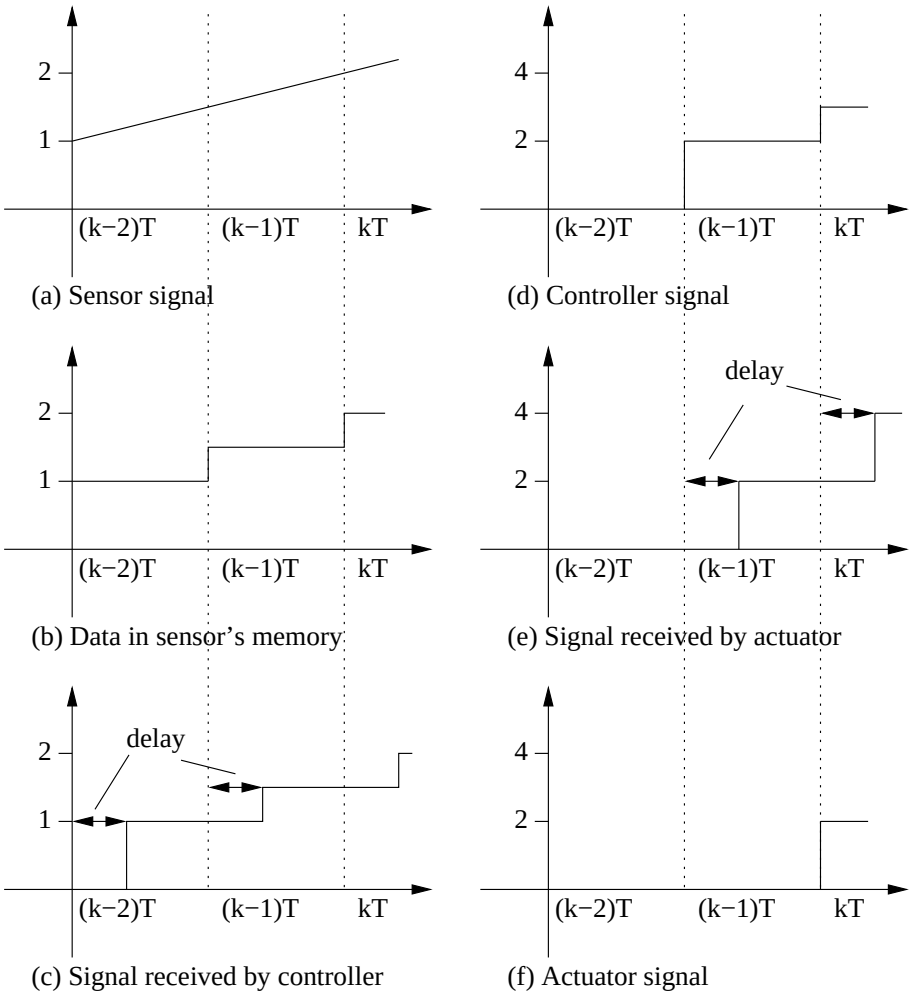


Figure 5.2: The timing diagram for sensors, controller, and actuators

a controller. However, this signal require some time (e.g. queuing and propagation time) to transmit to the controller. Therefore, the controller input buffer shows a delayed version of sampled sensor signal as shown in Fig. 5.2c. Since we assume the controller will amplify the input signal by two at every sampling time, the controller signal in its output buffer is shown in Fig. 5.2d. Again, the controller signal experience some delay before signal arrive to actuator. Therefore, signal received by actuator is a delayed version of controller signal as shown in Fig. 5.2e. Finally, the actuator changes its action at every sampling time and the actuator signal is shown in Fig.

5.2f. As a result, the actuator signal at time kT is equal to two multiply by the sensor signal at time $(k-2)T$. Hence, the closed loop system is experiencing two step delay. For more general case where delay may not be less than one sampling period can be characterized as follow:

$$\begin{aligned}(d_a - 1)T < a_i \leq d_a T \\ (d_s - 1)T < s_i \leq d_s T\end{aligned} \tag{5.3}$$

where d_s, $d_a \in \mathbb{Z}^+$ correspond to the number of step delay that sensors and actuators experience, respectively. Therefore, the closed loop system will experience $d_a + d_s$ steps delay.

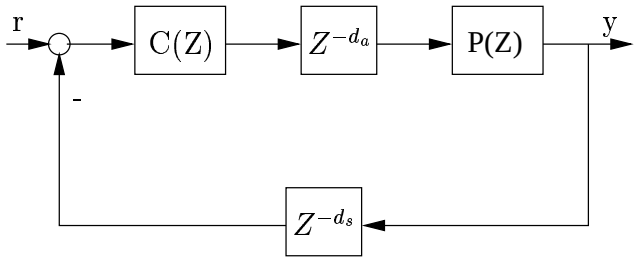


Figure 5.3: Discrete-time system with delay blocks

In the next two sections, we will discuss design of controller in discrete time because controllers are usually implemented on a digital computer for networked control systems. First, we assume d_s and d_a are constant, which is the case for system implemented in CAN and Profibus, and the delays are deterministic. Second, we will consider the case where d_s and d_a are random variables, which is the case for systems implemented in Ethernet, and the delays are non-deterministic. Before we start our discussion on controller design, we have to understand under what conditions a time delay system can be controlled and observed.

5.2 Controllability and Observability

Consider the linear system

$$\begin{aligned} x_1(k+1) &= Ax_1(k) + Bu_1(k) \\ y_1(k) &= Cx_1(k) \end{aligned} \tag{5.4}$$

where $x_1 \in \mathbb{R}^n$, $u_1 \in \mathbb{R}^m$ and $y_1 \in \mathbb{R}^r$.

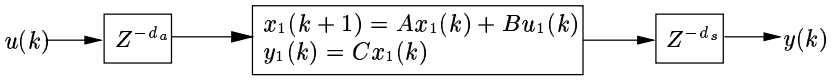
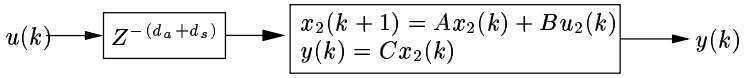


Figure 5.4: System with delay input and output

Theorem 3. *The delay system (Fig. 5.4) is controllable and observable iff the nominal system is controllable and observable*

Proof. • Controllability:

From the input-output point of view for the overall system, we can interchange the order of system and still preserve the same structure. We define $x_2(k) = x_1(k - d_s)$ and $u_2(k) = u_1(k - d_s)$. If we rearrange the systems so that the output delays are absorbed by the input delays.



Now, let us define a new state variable as follow:

$$x(k) = \left[x_2(k)^T u_2(k)^T u_2(k+1)^T \ \ldots \ u_2(k + d_s + d_a - 1)^T\right]^T$$

As a result. the combined system state equation can be written as follow:

$$x(k+1) = \begin{bmatrix} A & B & 0 \\ 0 & 0 & I_{m\times(d_a+d_s-1)} \\ 0 & 0 & 0 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 0 \\ I_m \end{bmatrix} u(k)$$

$$y(k) = \begin{bmatrix} C & 0 & 0 \end{bmatrix} x(k)$$

where $x \in \mathbb{R}^{n+m(d_a+d_s)}$, $u \in \mathbb{R}^m$ and $y \in \mathbb{R}^r$.

We define the controllability matrix for the non-delay system as follow:

$$\mathcal{C}(A,B) = \begin{bmatrix} B & AB & \dots & A^{n-1}B \end{bmatrix} \tag{5.5}$$

So, the system is controllable if and only if the controllability matrix is full rank. Let $A_{combine} = \left[\begin{smallmatrix} A & B & 0 \\ 0 & 0 & I_{m\times(d_a+d_s-1)} \\ 0 & 0 & 0 \end{smallmatrix}\right]$ and $B_{combine} = \left[\begin{smallmatrix} 0 \\ 0 \\ I_m \end{smallmatrix}\right]$ and we can define the controllable matrix for the delayed system as follow:

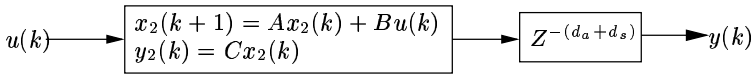
$$\mathcal{C}(A_{combine}, B_{combine}) = \begin{bmatrix} 0 & 0 & 0 & B & AB & \cdot & A^{n-1}B & \dots & A^{n+(m-1)(d_a+d_s)}B \\ 0 & 0 & I_m & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & \cdot & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ I_m & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & \mathcal{C}(A,B) & A^nB & \dots & A^{n+(m-1)(d_a+d_s)}B \\ 0 & 0 & I_m & 0 & 0 & \dots & 0 \\ 0 & \cdot & 0 & 0 & 0 & \dots & 0 \\ I_m & 0 & 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$

Hence, if $\mathcal{C}(A,B)$ is full rank then $\mathcal{C}(A_{combine}, B_{combine})$ is full rank and vice versa.

- Observability

  Similarly, we define $x_2(k) = x_1(k+d_a)$ and $y_2(k) = y_1(k+d_a)$. We can rearrange the overall system as follow:



  Now, let us define a new state variable as follow:

$$x(k) = \left[x_2(k)^T y_2(k-1)^T y_2(k-2)^T \ \ldots \ y_2(k-d_s-d_a)^T\right]^T$$

  As a result. the combined system state equation can be written as follow:

$$x(k+1) = \begin{bmatrix} A & 0 & 0 \\ C & 0 & 0 \\ 0 & I_{r\times(d_a+d_s-1)} & 0 \end{bmatrix} x(k) + \begin{bmatrix} B \\ 0 \\ 0 \end{bmatrix} u(k)$$

$$y(k) = \begin{bmatrix} 0 & 0 & I_r \end{bmatrix} x(k)$$

  where $x \in \mathbb{R}^{n+r(d_a+d_s)}$, $u \in \mathbb{R}^m$ and $y \in \mathbb{R}^r$.

  We define the observability matrix for the non-delay system as follow:

$$\mathcal{O}(A,C) = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{5.6}$$

  So, the system is observable if and only if the observability matrix is full rank.

Let $A_{combine} = \begin{bmatrix} A & 0 & 0 \\ C & 0 & 0 \\ 0 & I_{r\times(d_a+d_s-1)} & 0 \end{bmatrix}$ and $C_{combine} = \begin{bmatrix} 0 & 0 & I_r \end{bmatrix}$ and we define the observable matrix for the delayed system as follow:

$$\mathcal{O}(A_{combine}, C_{combine}) = \begin{bmatrix} 0 & 0 & 0 & I_r \\ 0 & 0 & \cdot & 0 \\ 0 & I_r & 0 & 0 \\ C & 0 & 0 & 0 \\ CA & 0 & 0 & 0 \\ \vdots & 0 & 0 & 0 \\ CA^{n-1} & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ CA^{n+(r-1)(d_a+d_s)} & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & I_r \\ 0 & 0 & \cdot & 0 \\ 0 & I_r & 0 & 0 \\ \mathcal{O}(A,C) & 0 & 0 & 0 \\ CA^{n} & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ CA^{n+(r-1)(d_a+d_s)} & 0 & 0 & 0 \end{bmatrix}$$

Hence, if $\mathcal{O}(A, C)$ is full rank then $\mathcal{O}(A_{combine}, C_{combine})$ is full rank and vice versa.

□

5.3 Controller Design with Constant Delay

Control of system with delay in the feedback loop has been a challenge problem for the last 50 years. The most common type of controller, which compensates delay, is known as the Smith predictor [19, 33]. A discrete time version [27] of Smith predictor is shown in Fig. 5.5. The Smith predictor consists of a primary controller ($C(z)$) and models of the controlled process with ($z^{-d_s}P(z)$) and without ($P(z)$) time delay. When the models and the process match exactly, the compensator removes the time delay factor from the characteristic equation.

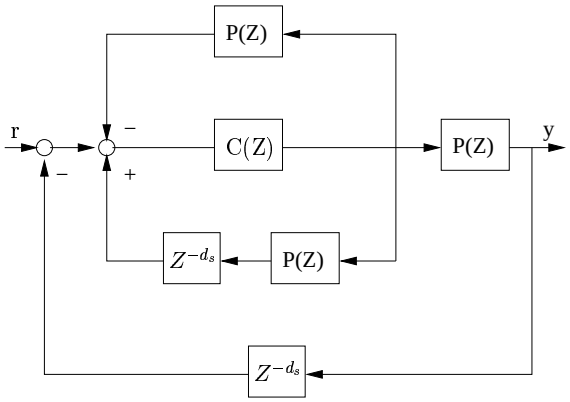


Figure 5.5: Smith Predictor

However, there is a major drawback due to the internal instability of the prediction that the controller will fail to stabilize unstable system. It will be more obvious as we explicitly write down the equation for Smith predictor, as follow:

$$C_{smith}(z) = \frac{C(z)}{1 + C(z)P(z)(1 - z^{-d_s})} \tag{5.7}$$

If $P(z)$ has an unstable pole, then C_{smith} will have the same unstable zero. Hence, the closed loop system becomes unstable as there is unstable pole-zero cancellation. Although Smith predictor cannot stabilize an unstable system, its rather simple design methodology attracts many researchers to develop different types of controllers based

on the same framework. In [21, 20], an improved model of Smith predictor was developed and we will discuss their ideas and results.

5.3.1 Continuous Resetting Smith Predictor

In this section, we consider observable liner multi-variable system with nominal delay h in the input described by

$$\begin{aligned} \dot{x}(t) &= A_c x(t) + B_c u(t-h) \\ y(t) &= C_c x(t) \end{aligned} \tag{5.8}$$

where $A_c \in \mathbb{R}^{n\times n}$, $B_c \in \mathbb{R}^{n\times m}$, and $C_c \in \mathbb{R}^{p\times m}$. A prediction $x_p(t)$ for the variable x(t+h), is given by

$$x_p(t) = e^{A_c h}x(t) + \int_t^{t+h} e^{A_c(t+h-\sigma)}B_c u(\sigma - h)d\sigma \tag{5.9}$$

Defining $\tau = \sigma - h$, $x_p(t)$ can be rewritten as

$$x_p(t) = e^{A_c h}x(t) + \int_{t-h}^{t} e^{A_c(t-\tau)}B_c u(\tau)d\tau \tag{5.10}$$

The above depends only on past and present values of $x(t)$, and $u(t)$. Thus, $x_p(t)$ is available at time t. However, as shown in [31], when the integral is calculated with a constant step method, such implementation produces an unstable behavior if the control law is itself unstable, whatever the precision of the integration method.

The purpose of [20], is to allow the Smith predictor to control unstable systems, without losing its main advantage, namely, the simple design procedure. In [20], the authors developed a Smith predictor which refreshes (resets and updates) its initial condition periodically. The calculator block computes an estimate of the state $x(t_i+h)$ for $i = 1, ..., k, k+1, ...$ The estimated value $x(t_i+h)$ is used as the initial value of the

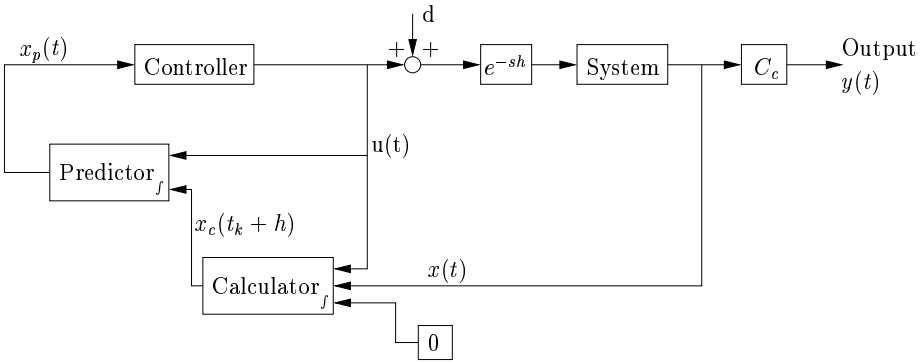


Figure 5.6: Continuous Resetting Smith Predictor

predictor which will deliver a "continuous" estimate of the state $x(t+h)$. The initial condition of the model is periodically reset to the value of the state of the system.

Calculator

During the time interval $[t_k - h, t_k)$ we compute the value of the integral

$$\int_{t_k-h}^{t_k} e^{A_c(t-\tau)} B_c u(\tau) d\tau \tag{5.11}$$

as a solution to the differential equation

$$\dot{z}(t) = A_c z(t) + B_c u(t) \tag{5.12}$$

with zero initial conditions at time $t_k - h$. At time t_k this integral, along with the measured value of the state of the system allows the computation of

$$\begin{aligned} x_c(t_k + h) &= e^{A_c h} x(t_k) + \int_{t_k}^{t_h+h} e^{A(t_k+h-\sigma)} B_c u(\sigma - h) d\sigma \\ &= e^{A_c h} x(t_k) + \int_{t_k-h}^{t_k} e^{A_c(t-\tau)} B_c u(\tau) d\tau \end{aligned} \tag{5.13}$$

where $\tau = \sigma - h$ and Eqn. 5.13 is an estimate of $x(t_k + h)$ computed at time t_k. Next, at time t_k, $k = 1, 2, ...$ the predictor is initialized with the computed value $x_c(t_k + h)$.

Predictor

At time t_k, $k = 1, 2, ...$ the predictor is initialized with the computed value $x_c(t_k + h)$. The prediction $x_p(t)$ of the state $x(t + h)$ is given as the solution to

$$\begin{aligned} \dot{x}_p(t) &= A_c x_p(t) + B_c u(t) \quad \text{for } t \in [t_k, t_{k+1}), \\ x_p(t_k) &= x_c(t_k + h) \end{aligned}$$

Equivalently, $x_p(t)$ is given by

$$x_p(t) = e^{A_c(t-t_k)} x_c(t_k + h) + \int_{t_k}^{t} e^{A_c(t-\tau)} B_c u(\tau) d\tau, \quad t \in [t_k, t_{k+1}). \tag{5.14}$$

Control law

The control law is a static state feedback

$$u(t) = -K_c x_p(t); \quad t \leq 0 \tag{5.15}$$

that stabilizes the system for the nominal parameters.

We will skip the stability analysis in this paper because we will implement control algorithm in digital computer. The brief summary of continuous resetting Smith predictor is introduce the idea of how to interpret the design. In short, the predictor is governed by the stable difference equation

$$x_p(t_{k+1}) = e^{(A_c - B_c K_c)(t_{k+1} - t_k)} x_p(t_k) \quad \text{for } k = 1, 2, ... \tag{5.16}$$

where K_c is the static controller gain that stabilize the non-delay closed loop system. For those who are interested in the continuous dynamics can study the stability and robust analysis given in [21].

5.3.2 Discrete Resetting Smith Predictor

For the discretization of the scheme, the sampling period, T, is selected such that the design time delay, h, is an integer multiple of T. It is well known that in the discrete framework the sampling period must be small enough so that it meets the requirements inherent to the discretization process. Since the sampling period is a design parameter it can be chosen so that it fulfills both requirements, namely, $nT = h$, where n is an integer.

The discrete resetting Smith predictor is obtained by choosing the resetting time of the previous section so that it coincides with the sampling time. A zero order hold is used in the control action. Consider Eqn. 5.14 at time $t = kT$, $k = 0, 1, 2, ...$, then

$$x_p(kT) = e^{A_c(nT)}x(kT) + \int_{kT-nT}^{kT} e^{A_c(kT-\tau)}B_c u(\tau)d\tau \tag{5.17}$$

Let $kT - \tau = \zeta$, it follows that

$$x_p(kT) = e^{A_c(nT)}x(kT) + \int_{0}^{nT} e^{A_c\zeta}B_c u(kT-\zeta)d\zeta \tag{5.18}$$

Because of the zero order hold, $u(kT-\zeta)$ is piecewise constant over each intersampling interval, hence

$$x_p(kT) = e^{A_c(nT)}x(kT) + \sum_{j=1}^{n}\int_{(j-1)T}^{jT} e^{A_c\zeta}B_c d\zeta u(kT-jT) \tag{5.19}$$

Define $\sigma = \zeta - (j-1)T$ for $j = 1, ..., n$, in the following equations

$$\begin{aligned} \int_{(j-1)T}^{jT} e^{A_c\zeta} B_c d\zeta &= \int_0^T e^{A_c[(j-1)T+\sigma]} B_c d\sigma \\ &= e^{A_c(j-1)T} \int_0^T e^{A_c\sigma} B_c d\sigma \end{aligned} \tag{5.20}$$

then

$$\begin{aligned} x_p(kT) &= e^{A_c(nT)} x(kT) + \sum_{j=1}^{n} \int_{(j-1)T}^{jT} e^{A_c\zeta} B_c d\zeta u(kT - jT) \\ &= e^{A_c(nT)} x(kT) + \sum_{j=1}^{n} \phi^{j-1} \Gamma u(kT - jT) \\ &= \phi^n x(kT) + \Phi(z) u(kT) \end{aligned} \tag{5.21}$$

where $z^{-1}u(kT) = u(kT - T)$, $\phi = e^{A_cT}$, $\Gamma = \int_0^T e^{A_c\sigma} B_c d\sigma$ and, $\Phi(z) = \phi^{n-1}\Gamma z^{-n} + ... + \phi\Gamma z^{-2} + \Gamma z^{-1}$. Again, due to the delayed nature of the input, the prediction at the sampling time $kT + nT$, depends only on $x(kT)$ and of values of the input at kT and previous sampling instants. Hence this prediction is available at time kT. The overall control scheme is then:

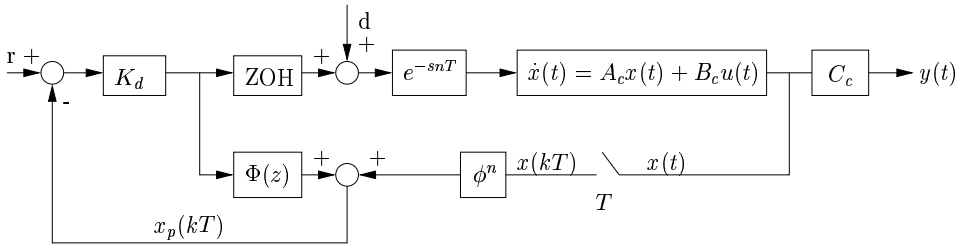


Figure 5.7: Discrete resetting Smith Predictor (DRSP)

The design of this predictor is based on the continuous time resetting Smith predictor that has been explained in section 5.3.1. DRSP is a predictor which refreshes (resets and updates) its initial condition periodically. This refresh mechanism avoids drifting of the state estimation error. The predicted states can then be used by

constant gain state feedback controller (K_d) to control the delayed system.

Stability Analysis

The reference to output transfer function is

$$\frac{x(z)}{r(z)} = \frac{K_d P^*(z) z^{-n^*}}{I + K_d \Phi(z) + K_d \phi^n P^*(z) z^{-n^*}} \tag{5.22}$$

where $P(z)$ is the model transfer function and $*$ corresponds to the real behavior. Since the denominator of the transfer function contain $\Phi(z)$ term, which is directly related to the system, we can find the relationship of $\Phi(z)$ and $P(z)$, as follows:

$$\begin{aligned}
\Phi(z) &= \phi^{n-1}\Gamma z^{-n} + ... + \phi\Gamma z^{-2} + \Gamma z^{-1} \\
&= \left(\phi^{n-1} z^{-(n-1)} + ... + I\right) \Gamma z^{-1} \\
&= \left[I - \left(\phi z^{-1}\right)^n\right] \left(I - \phi z^{-1}\right)^{-1} \Gamma z^{-1} \\
&= \left[I - \left(\phi z^{-1}\right)^n\right] (zI - \phi)^{-1} \Gamma \\
&= \left[I - \left(\phi z^{-1}\right)^n\right] P(z)
\end{aligned} \tag{5.23}$$

Substitute Eqn. 5.23 into Eqn. 5.22, we get

$$\frac{x(z)}{r(z)} = \frac{K_d P^*(z) z^{-n^*}}{I + K_d \left[I - (\phi z^{-1})^n\right] P(z) + K_d \phi^n P^*(z) z^{-n^*}} \tag{5.24}$$

When there is no mismatch in the parameters, the delay of the process ($n = n^*$), and the design model ($P(z) = P^*(z)$), this transfer function simplifies to

$$\frac{x(z)}{r(z)} = \frac{K_d P(z) z^{-n}}{I + K_d P(z)} \tag{5.25}$$

Therefore, if the controller gain K_d is designed so that the closed loop system with no delay has a stable closed loop polynomial, it follows that the closed loop system

resulting from the scheme of Fig. 5.7 has the same characteristic polynomial. Hence, it is stable.

Robustness Analysis

Theorem 4. *(Theorem 1 of [20])*

Consider the discretization of the process in closed loop with the control law, K_d, Eqn. 5.21, and assume that the controller is designed so that under ideal circumstances, the closed loop is stable. Assume also that the real process and design model differ. Then, if the condition

$$|K_d\phi^n \left\{P^*(z)z^{n^*-n} - P(z)\right\} z^n| < |I + K_d\Phi(z) + K_d\phi^n P(z)z^n| \tag{5.26}$$

holds for all z on the unit circle, then the closed loop remains stable.

5.3.3 Example: Magnetically Levitated Ball (MLB)

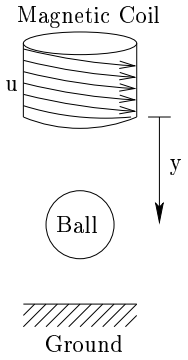


Figure 5.8: Magnetically levitated ball

We illustrate the discrete resetting Smith predictor scheme using a magnetic levitation system as shown in Fig. 5.13. The simplified differential equation for the

system is given by

$$\ddot{y} = g - k\left(\frac{u^2}{y^2}\right) \tag{5.27}$$

where u the control input is the current through the electromagnet, y the output is the vertical displacement of the ball from the magnet, g is the gravitational acceleration, and k is a constant determined by the physical dimensions and material of the system. For this example, $g = 9.81$, and $k = 0.01$, the sampling time is $T = 0.05$ second. Assume the system is linearized at $y = 0.5$, and there is 1-step delay on both sensor output and controller output. The discretized system is giving as follows:

$$x(k+1) = \begin{bmatrix} 1.0495 & 0.0508 \\ 1.9942 & 1.0495 \end{bmatrix} x(k) + \begin{bmatrix} -0.0016 \\ -0.0637 \end{bmatrix} u(k) \tag{5.28}$$

First, we design an LQR controller using weighting matrices $Q_x = I_2$ for the states and $R_u = 1$ for the control signal. We get

$$K_d = \begin{bmatrix} -54.1949 & -8.6815 \end{bmatrix} \tag{5.29}$$

and poles at 0.7529 and 0.7076. Since we expect the overall system will experience 2-step delay, we design the discrete resetting Smith predictor that control the system with 2-step delay that is, $n = 2$. The equation for state prediction is written as follows:

$$\begin{aligned} x_p(k) &= e^{2A_cT}x(k) + \Phi(z)u(k) \\ &= e^{2A_cT}x(k) + \Gamma u(k-1) + \phi\Gamma u(k-2) \end{aligned} \tag{5.30}$$

Fig. 5.9 is a simulation result uses continuous dynamics with initial condition of $y(0) = 1.3$ and $\dot{y}(0) = 0.5$. The figure shows that the closed loop system is stable.

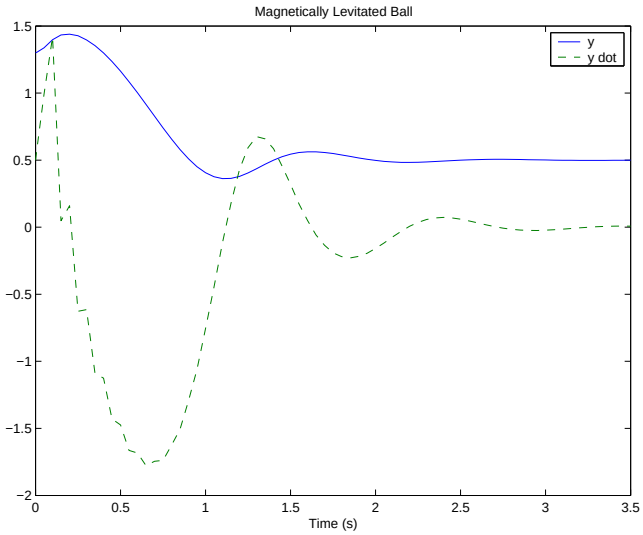


Figure 5.9: Transient Response

5.4 Mean Square Stability

In the previous section, we have discussed the control design methodology for system with constant delay. However, if a system is implemented through a communication network, such as Ethernet, then the system may not experience constant delay all the time. Therefore, we should find a way to characterize stability for such a stochastic system. Furthermore, we will only consider a special class of stochastic systems that is a Markovian jump system. Discrete time linear jump systems are systems which are subject to abrupt changes in their parameters. These abrupt changes can be modeled by a Markov chain with a finite state-space. So, each mode of a jump system corresponds to a particular parameter setting for a system at a time instance.

We consider jump linear systems described by

$$x(k+1) = \widehat{A}_{r(k)}x(k) \tag{5.31}$$

where $x \in \mathbb{R}^n$ is the x-process state, and $\widehat{A}_{r(k)} \in \{\widehat{A}_0, ..., \widehat{A}_d\}$ is appropriately dimensioned matrices. Here, $k \in Z^+$ is the time index and $r(k)$ is a discrete-time

Markov process which takes values in the finite set $\mathbf{D}= \{1, 2, ..., d\}$. It is a finite state discrete-time Markov chain, with transition probability matrix

$$Pr\{r(k+1) = j | r(k) = i\} = \{p_{ij}\} \in \mathbb{R}^{D\times D} \tag{5.32}$$

where $i, j = 1, 2, ..., D$.

Definition 7. (Mean Square Stable[6]) *The model (5.31) is mean square stable (MSS) if for any initial condition $x(0)$ there exist $q \in \mathbb{R}^n$ and $Q \in \mathbb{R}^{n\times n}$ independent of $x(0)$ such that:*

1. $\| E[x(k)] - q \| \to 0$ *as* $k \to \infty$
2. $\| E[x(k)x(k)^T] - Q \| \to 0$ *as* $k \to \infty$

Define the mode indicator function:

$$I_i(k) = \begin{cases} I, & \text{if } r(k) = i \\ 0, & \text{if } r(k) \neq i \end{cases}$$

There are two ways to define the mode-dependent covariance matrices [6, 9]

$$(a) \qquad M_i^a(k) = E\left[x(k)x(k)^T I_i(k)\right] \tag{5.33}$$

$$(b) \qquad M_i^b(k) = E\left[x(k)x(k)^T I_i(k-1)\right] \tag{5.34}$$

They satisfy the linear recursions

$$M_i^a(k+1) = \sum_{j=0}^{d} p_{ji} \widehat{A}_j M_j^a(k) \widehat{A}_j^T \tag{5.35}$$

$$M_i^b(k+1) = \widehat{A}_i \left(\sum_{j=0}^{d} p_{ji} M_j^b(k) \right) \widehat{A}_i^T \tag{5.36}$$

respectively. All these equations hold for $i = 0, 1, ..., d$. The mode-independent covariance matrix is

$$M(k) = E\left[x(k)x(k)^T\right] = \sum_{i=0}^{d} M_i^a(k) = \sum_{i=0}^{d} M_i^b(k) \tag{5.37}$$

The system is mean square stable if $\lim_{k\to\infty} M(k) = 0$ regardless of $x(0)$. A necessary and sufficient condition for a jump linear system to be mean square stable is

$$\lim_{k\to\infty} M_i^a(k) = 0 \quad \lim_{k\to\infty} M_i^b(k) = 0, \quad i = 0, 1, ..., d \tag{5.38}$$

Theorem 5. *(Theorem 1 of [6]) Let* $\mathbb{A} = \left(P^T \otimes I_{n^2}\right) diag(\widehat{A}_{r(k)} \otimes \widehat{A}_{r(k)})$. *The system 5.31 is mean square stable if and only if the spectral radius of* $\mathbb{A}$ *is less than 1, where* $\otimes$ *denotes the matrix Kronecker product [11].*

This theorem is normally used to check the mean square stability of a system and it is rather difficult to do controller design through this expression. For design purposes, one may want to use the next theorem.

Theorem 6. *(Theorem 2 of [6]) The mean square stability of system 5.31 is equivalent to the existence of symmetric positive definite matrices* Q_0, Q_1, ..., Q_d *satisfying*

any one of the following 4 conditions:

$$
\begin{aligned}
&1. \quad \widehat{A}_i \left(\sum_{j=0}^{d} p_{ji} Q_j \right) \widehat{A}_i^T < Q_i, \qquad i = 0, ..., d \\
&2. \quad \widehat{A}_j^T \left(\sum_{i=0}^{d} p_{ji} Q_i \right) \widehat{A}_j < Q_j, \qquad j = 0, ..., d \\
&3. \quad \sum_{j=0}^{d} p_{ji} \widehat{A}_j Q_j \widehat{A}_j^T < Q_i, \qquad i = 0, ..., d \\
&4. \quad \sum_{i=0}^{d} p_{ji} \widehat{A}_i Q_i \widehat{A}_i^T < Q_j, \qquad j = 0, ..., d
\end{aligned} \tag{5.39}
$$

As we mentioned in Section 5.3.3, the normal sense of stabilizability does not apply to Markovian jump system. Therefore, we have to redefine the stabilizability and detectability for Markovian jump system which has the following form:

$$
\begin{aligned}
x(k+1) &= A_{r(k)} x(k) + B_{r(k)} u(k) \\
y(k) &= C_{r(k)} x(k)
\end{aligned} \tag{5.40}
$$

Definition 8. (Mean square stabilizability) *Consider system 5.40. We say that (A,B) is mean square stabilizable if there exists* $\{K_0, ..., K_d \mid K_i \in \mathbb{R}^{m,n} \text{ for } i = 0, ..., d\}$ *for which the system* $z(k+1) = \left(A_{r(k)} + B_{r(k)} K_{r(k)}\right) z(k)$, $z(0) = z_0$, $k \geq 0$, *is mean square stable.*

Definition 9. (Mean square detectability) *Consider system 5.40. We say that (A,C) is mean square detectable if there exists* $\{L_0, ..., L_d \mid L_i \in \mathbb{R}^{n,r} \text{ for } i = 0, ..., d\}$ *for which the system* $z(k+1) = \left(A_{r(k)} + L_{r(k)} C_{r(k)}\right) z(k)$, $z(0) = z_0$, $k \geq 0$, *is mean square stable.*

One interesting property is that stability of every mode is neither sufficient nor necessary for mean square stability of the jump system [13].

5.5 Controller design with random delay

In section 5.3, we have discussed the control design methodology for system with constant delay. However, if a system is implemented through a communication network, such as Ethernet, then the system may not experience constant delay all the time. In addition, we cannot use the usual control design methodology, like the Smith predictor, for system with random delay because it does not guarantee stability for such system. In section 5.4, we have presented the notion of mean square stability for Markovian jump systems. This notion enables us to design controller for systems with random delay. First, we will model the random delay system with state feedback control as a jump system[45]. Secondly, we will model the delay sequence as a Markov process. Thirdly, we will derive a more general model for Markovian jump system by using dynamic controller. Finally, we will provide a controller design algorithm and an illustrative example.

5.5.1 Delayed State Deedback Model

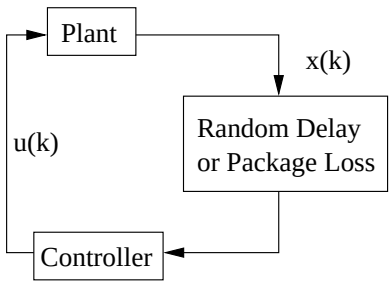


Figure 5.10: Networked Control System

First consider the simple system setup in Fig. 5.10. The discrete-time linear time-invariant plant model is

$$x(k+1) = Ax(k) + Bu(k) \tag{5.41}$$

where $x(k) \in \mathbb{R}^n$, $u(k) \in \mathbb{R}^m$. It is assumed that there are random but bounded delays from the sensor to the controller. The mode-dependent switching state feedback control law is

$$u(k) = K_{r_s(k)} x(k - r_s(k)) \tag{5.42}$$

where $\{r_s(k)\}$ is a bounded random integer sequence with $0 \leq r_s(k) \leq d_s < \infty$, and d_s is the finite integer delay bound. Let us introduce a new state variable, that contain all the state information of the linear system with different delay, as follow:

$$\tilde{x}(k) = \begin{bmatrix} x(k)^T & x(k-1)^T & \dots & x(k-d_s)^T \end{bmatrix}^T$$

where $\tilde{x}(k) \in \mathbb{R}^{(d_s+1)n}$, then the closed loop system is

$$\tilde{x}(k+1) = \left(\tilde{A} + \tilde{B} K_{r_s(k)} \tilde{C}_{r_s(k)} \right) \tilde{x}(k) \tag{5.43}$$

where

$$\tilde{A} = \begin{bmatrix} A & 0 & \dots & 0 & 0 \\ I & 0 & \dots & 0 & 0 \\ 0 & I & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & I & 0 \end{bmatrix} \qquad \tilde{B} = \begin{bmatrix} B \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\tilde{C}_{r_s(k)} = \begin{bmatrix} 0 & \dots & 0 & I & 0 & \dots & 0 \end{bmatrix}$$

and $\tilde{C}_{r_s(k)}$ has all elements being zero except for the $r_s(k)$th block being an identity matrix. Eqn. 5.43 corresponds to a discrete-time jump linear system. Notice that these equations are in the form of an output feedback control problem, even if a state feedback control law, Eqn. 5.42, was intended for the original system, Eqn. 5.41. However, we haven't properly characterize the random sequence $r_s(k)$ and this will be formulated in the next section.

5.5.2 Markovian Jump System

One of the difficulties with this approach is how to model $r_s(k)$ sequence. One way is to model the transitions of the random delays $r_s(k)$ as a finite state Markov process [25, 45]. In general, system networks (such as fully-switched ethernet) are under control environment which is not totally random with the future system characteristics dependent on the current state. Therefore, it is a reasonable assumption to model the delay sequence as a Markov process. In this case we have

$$Prob\{r_s(k+1) = j | r_s(k) = i\} = p_{ij} \tag{5.44}$$

where $0 \leq i, j \leq d_s$. This model is quite general, communication package loss in the network can be included naturally as explained below. The assumption here is that the controller will always use the most recent data. Thus, if we have $x(k - r_s(k))$ at step k, but there is no new information coming at step $k + 1$ because data are lost or there is a longer delay, then we at least have $x(k - r_s(k))$ available for controller. So in our model of the system in Fig. 5.10, the delay $r_s(k)$ can increase at most by 1 each step, and we constrain

$$Prob\{r_s(k+1) > r_s(k) + 1\} = 0 \tag{5.45}$$

However, the delay $r_s(k)$ can decrease as many steps as possible. Decrement of $r_s(k)$ models communication packet loss in the network, or disregarding old data if we have newer data coming at the same time. Hence, the structured transition probability matrix is

$$P_s = \begin{bmatrix} p_{00} & p_{01} & 0 & 0 & \dots & 0 \\ p_{10} & p_{11} & p_{12} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & p_{d_s-1 d_s} \\ p_{d_s 1} & p_{d_s 2} & p_{d_s 3} & p_{d_s 4} & \dots & p_{d_s d_s} \end{bmatrix} \tag{5.46}$$

where $0 \leq p_{ij} \leq 1$ and $\sum_{j=0}^{d_s} p_{ij} = 1$. The diagonal elements are the probabilities of data coming in sequence with equal delays. The elements above the diagonal are the probabilities of encountering longer delays, and the elements below the diagonal indicate packet loss or disregarding old data. Fig. 5.11 shows a four state transition diagram with such a structure, which clearly shows that we can jump from $r = 0$ to $r = 1$ and from all other states to $r = 0$, but we cannot jump directly from $r = 0$ to $r = 2$ or $r = 3$.

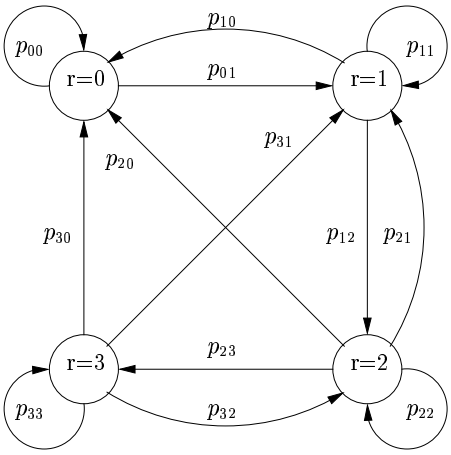


Figure 5.11: Markovian Jump States

5.5.3 Dynamic Output Feedback

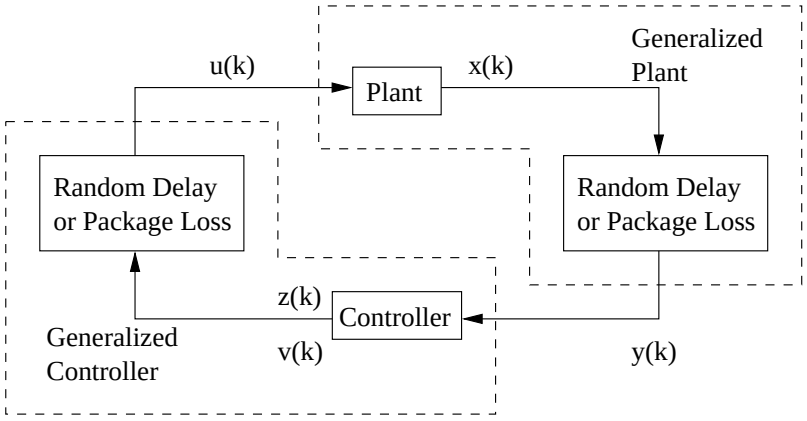


Figure 5.12: Networked Control System: the general case

Next we consider a more general model with a dynamic feedback controller, Fig 5.12,

$$\begin{aligned} z(k+1) &= Fz(k) + Gy(k) \\ v(k) &= Hz(k) + Jy(k) \end{aligned} \tag{5.47}$$

where $y(k) = Cx(k - r_s(k))$ and $u(k) = v(k - r_a(k))$. In this case we use a mode-independent controller to simplify the notation which means the controller's parameters will not change, e.g. $F = F_0 = ... = F_{r_s(k)}$ for $\forall k \in Z^+$. More importantly, because it is hard to predict the delays from the controller to the actuator at the time the control signal is calculated, the mode-independent controller is probably the most relevant for this application. To proceed, augment the controller state variable

$$\tilde{z}(k) = \begin{bmatrix} z(k)^T & v(k-1)^T & \dots & v(k-d_a)^T \end{bmatrix}^T$$

where d_a is the bound for the random delays $r_a(k)$ from the controller to the actuator. The generalized controller can be written as

$$\begin{aligned} \tilde{z}(k+1) &= \tilde{F}\tilde{z}(k) + \tilde{G}y(k) \\ u(k) &= \tilde{H}_{r_a(k)}\tilde{z}(k) + \tilde{K}_{r_a(k)}y(k) \end{aligned} \tag{5.48}$$

where

$$\tilde{F} = \begin{bmatrix} F & 0 & \dots & 0 & 0 \\ H & 0 & \dots & 0 & 0 \\ 0 & I & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & I & 0 \end{bmatrix} \qquad \tilde{G} = \begin{bmatrix} G \\ J \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\tilde{H}_{r_a(k)} = \begin{cases} \begin{bmatrix} H & 0 & \dots & 0 & 0 & \dots & 0 \end{bmatrix} & \text{if } r_a(k) = 0 \\ \begin{bmatrix} 0 & \dots & 0 & I & 0 & \dots & 0 \end{bmatrix} & \text{if } r_a(k) \neq 0 \end{cases}$$

$$\tilde{K}_{r_a(k)} = \begin{cases} J & \text{if } r_a(k) = 0 \\ 0 & \text{if } r_a(k) \neq 0 \end{cases}$$

The identity matrix, I, in $\tilde{H}_{r_a(k)}$ is positioned at the $r_a(k)^{th}$ entry of the row matrix. Here we use the control signal $v(k), ..., v(k - d_a)$ generated at different steps for analysis and design purpose only. After the controller, Eqn. 5.47, is designed using the generalized model, we need not to store them in real-time control applications. The transition probability matrix P_a has the same structure as P_s in Eqn. 5.46.

For a discrete-time linear time-invarient plant model

$$x(k+1) = Ax(k) + Bu(k)$$

where $x(k) \in \mathbb{R}^n$ and $u(k) \in \mathbb{R}^m$, we can transform this model into a generalized plant model that include output delay. Let us introduce a new state variable, that contain all the state information of the linear system with different delay, as follow:

$$\tilde{x}(k) = \begin{bmatrix} x(k)^T & x(k-1)^T & \dots & x(k-d_s)^T \end{bmatrix}^T$$

where $\tilde{x} \in \mathbb{R}^{(d_s+1)n}$, then the generalized plant model can be written as follow:

$$\begin{aligned} \tilde{x}(k+1) &= \tilde{A}\tilde{x}(k) + \tilde{B}u(k) \\ y(k) &= C\tilde{C}_{r_s(k)}\tilde{x}(k) \end{aligned} \tag{5.49}$$

where

$$\tilde{A} = \begin{bmatrix} A & 0 & \dots & 0 & 0 \\ I & 0 & \dots & 0 & 0 \\ 0 & I & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & I & 0 \end{bmatrix} \qquad \tilde{B} = \begin{bmatrix} B \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\tilde{C}_{r_s(k)} = \begin{bmatrix} 0 & \dots & 0 & I & 0 & \dots & 0 \end{bmatrix}$$

After defining the generalized plant and controller model, the next step is to combined these two models together as a generalized close loop system. First, we will combine Eqn. 5.48 and Eqn. 5.49 together and the resulting state equations are given as follow:

$$\begin{aligned} \tilde{x}(k+1) &= \tilde{A}\tilde{x}(k) + \tilde{B}\tilde{H}_{r_a(k)}\tilde{z}(k) + \tilde{B}\tilde{K}_{r_a(k)}C\tilde{C}_{r_s(k)}\tilde{x}(k) \\ \tilde{z}(k+1) &= \tilde{F}\tilde{z}(k) + \tilde{G}C\tilde{C}_{r_s(k)}\tilde{z}(k) \end{aligned} \tag{5.50}$$

Secondly, by introducing a new state variable $\bar{x} = [\tilde{x}^T \quad \tilde{z}^T]^T$, we can write the generalized closed loop system dynamics as

$$\bar{x}(k+1) = \left(\bar{A} + \bar{B}\bar{K}_{r_a(k)}\bar{C}_{r_s(k)}\right)\bar{x}(k) \tag{5.51}$$

where

$$\bar{A} = \begin{bmatrix} \tilde{A} & 0 \\ 0 & 0 \end{bmatrix} \qquad \bar{B} = \begin{bmatrix} 0 & \tilde{B} \\ I & 0 \end{bmatrix}$$

$$\bar{C}_{r_s(k)} = \begin{bmatrix} 0 & I \\ C\tilde{C}_{r_s(k)} & 0 \end{bmatrix} \qquad \bar{K}_{r_a(k)} = \begin{bmatrix} \tilde{F} & \tilde{G} \\ \tilde{H}_{r_a(k)} & \tilde{K}_{r_a(k)} \end{bmatrix}$$

This general system is also a jump linear system. The Markovian jump parameter now becomes $r(k) = (r_s(k), r_a(k))$, and the transition probability matrix is $P = P_s \otimes P_a$, where $\otimes$ denotes the matrix Kronecker product [11]. So, both static and dynamic feedback with random delays can be formulated as discrete-time jump system control problems.

However, if the transition probability matrix P is only known to belong to the polytopes $\Pi = C_0\{P_1, P_2, ..., P_t\}$, then a necessary and sufficient condition for the jump system to be mean square stable for every $P \in \Pi$ is that Theorem 6 holds for every vertex P_i of the polytopes [9]

$$P = \left\{ \Pi = \sum_{k=1}^{L} \lambda_k \Pi^k \mid \lambda_k \geqslant 0, k = 1, ..., L, \sum_{k=1}^{L} \lambda_k = 1 \right\} \tag{5.52}$$

where $\Pi^k = (p_{ij}^k)_{1 \leq i,j, \leq N}$, $k = 1, ..., L$ are given transition probability matrices. The convex hull $C_0\{\mathbb{S}\}$ of a set $\mathbb{S}$ is the intersection of all convex sets containing $\mathbb{S}$. For example, if a closed loop system with transition probability matrix P_1 and P_2 is mean square stable, then the closed loop system with transition probability matrix $P_3 = \lambda_1 P_1 + \lambda_2 P_2$, where $\lambda_1 + \lambda_2 = 1$, is also mean square stable. We will demonstrate this idea in the next example.

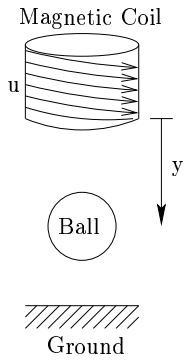


Figure 5.13: Magnetically levitated ball

Example

Let us consider the same magnetic levitation system as described in Section 5.3.3. The simplified differential equation for the system is given by

$$\ddot{y} = g - k\left(\frac{u^2}{y^2}\right) \tag{5.53}$$

where u the control input is the current through the electromagnet, y the output is the vertical displacement of the ball from the magnet, g is the gravitational acceleration, and k is a constant determined by the physical dimensions and material of the system. For this example, $g = 9.81$, and $k = 0.01$, the sampling time is $T = 0.05$ second. Assume the system is linearized at $y = 0.5$, and there is 1-step delay on both sensor output and controller output. The discretized system is giving as follows:

$$\begin{aligned} x(k+1) &= \begin{bmatrix} 1.0495 & 0.0508 \\ 1.9942 & 1.0495 \end{bmatrix} x(k) + \begin{bmatrix} -0.0016 \\ -0.0637 \end{bmatrix} u(k) \\ y(k) &= \begin{bmatrix} 1 & 0 \end{bmatrix} x(k - r_s(k)) \end{aligned} \tag{5.54}$$

For simplicity, we will assume no delay between controller and actuator and 0 or 1-step delay, $r_s(k) = \{0, 1\}$, could happen between sensor and controller. Indeed, the

delay factor is governed by Markov process and the transition probability matrix has the same structure as Eqn. 5.46. According to the design procedure in Section 5.5.4, we find a dynamic controller

$$\begin{aligned} z(k+1) &= \begin{bmatrix} 0.6681 & -0.0432 \\ -0.6361 & -0.0282 \end{bmatrix} z(k) + \begin{bmatrix} 0.4057 \\ -1.0942 \end{bmatrix} y(k) \\ u(k) &= \begin{bmatrix} -8.1992 & 4.1011 \end{bmatrix} z(k) + 52.8908 y(k) \end{aligned} \tag{5.55}$$

that can stabilize the system with two different transition probability matrix given as follow:

$$P_1 = \begin{bmatrix} 0.3450 & 0.6550 \\ 0.3450 & 0.6550 \end{bmatrix} \qquad P_2 = \begin{bmatrix} 0.6625 & 0.3375 \\ 0.9800 & 0.0200 \end{bmatrix} \tag{5.56}$$

According to Eqn. 5.52, the controller should able to achieve mean square stability for every transition probability matrix in the following form:

$$P_3 = \lambda_1 P_1 + (1 - \lambda_1) P_2 \tag{5.57}$$

where all entries in P_3 must be greater than or equal to zero. Fig. 5.14 is a simulation result uses nonlinear dynamics with initial condition of $y(0) = 1.3$ and $\dot{y}(0) = 0.5$. In the simulation, we set $\lambda_1 = 0.6$ and the results are consistent with the theory.

5.5.4 V-K iteration

In the previous section, we have transformed the random delay closed loop system into a Markovian jump linear system. Furthermore, we have a theorem that characterize the stability for this system. Therefore, as long as we find a set of controller's parameters that satisfies Theorem 6, the random delay control problem is solved.

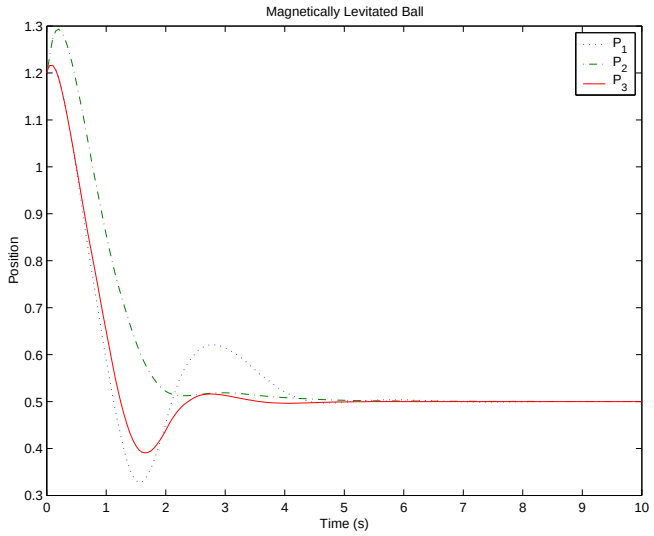


Figure 5.14: Transient Response with different transition probability

However, it is extremely difficult to find the set of parameters because it is a matter of trial and error. Fortunately, a systematic algorithm has been developed in [45] to solve this problem. Now we consider the problem of using output feedback control to stabilize the jump system, Eqn. 5.40. The closed loop system dynamics are

$$x(k+1) = \left(A_{r(k)} + B_{r(k)} K_{r(k)} C_{r(k)}\right) x(k) \tag{5.58}$$

which can represent the static feedback case, Eqn. 5.43, or the dynamic output feedback case, Eqn. 5.51, but is more general than these cases. Before we discuss the algorithm, we need to modify Eqn. 5.39 slightly by including a decay rate term. The decay rate is defined as the largest $\beta > 1$ such that $\lim_{k\to\infty} \beta^k M(k) = 0$. A lower bound of the decay rate $\beta = \frac{1}{\alpha}$ must satisfy the inequalities 5.39 by replacing Q_i or Q_j on the right hand side by αQ_i or αQ_j. The functionality of decay rate will become obvious as we go through the algorithm.

Since the four conditions in Theorem 6 are equivalent, we can use any one of them to describe our algorithm. For example, condition 4 in Eqn. 5.39 with decay rate

$\beta = \frac{1}{\alpha}$ for the closed loop system, Eqn. 5.58, becomes

$$\sum_{i=0}^{d} p_{ij} \widehat{A}_i^T Q_i \widehat{A}_i < \alpha Q_j \tag{5.59}$$

where we define $\widehat{A}_i = A_i + B_i K_i C_i$. Using Schur complements [2], Eqn. 5.59 can be written in an equivalent form

$$\begin{bmatrix} \alpha Q_j & \widehat{A}_0^T Q_0 & \dots & \widehat{A}_d^T Q_d \\ Q_0 \widehat{A}_0 & p_{j0}^{-1} Q_0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ Q_d \widehat{A}_d & 0 & \dots & p_{jd}^{-1} Q_d \end{bmatrix} > 0 \tag{5.60}$$

which must be true for $j = 0, 1, ..., d$ (Q>0 means Q is positive definite). These are coupled bilinear matrix inequalities (BMI) in the variables α, K_i, and Q_i, $i = 0, 1, ..., d$, and every K_i and Q_i appear in all of the $d+1$ inequalities. Of course, if we fix α and the K_i's, then these equations are LMI's in the Q_i's, and vice versa.

A lower bound for the decay rate $\beta = \frac{1}{\alpha}$ can be found by solving the following optimization problem

$$\begin{array}{ll} \text{minimize} & \alpha \\ \text{subject to} & \text{Eqn. 5.60 for } j = 0, 1, ..., d \\ & \text{and } Q_0 > 0, ..., Q_d > 0 \end{array} \tag{5.61}$$

If we fix the K_i's, this corresponds to a generalized eigenvalue problem. The generalized eigenvalue problem (GEVP) is to minimize the maximum generalized eigenvalue of a pair of matrices that depend affinely on a variable, subject to an LMI constraint.

The general form of a GEVP[2] is:

$$\begin{aligned} \text{minimize} \quad & \alpha \\ \text{subject to} \quad & \alpha B(x) - A(x) > 0, B(x) > 0 \end{aligned} \tag{5.62}$$

where A, and B are symmetric matrices that are affine functions of x. We can express this as

$$\begin{aligned} \text{minimize} \quad & \alpha_{\max}\left(A(x), B(x)\right) \\ \text{subject to} \quad & B(x) > 0 \end{aligned} \tag{5.63}$$

where $\alpha_{\max}(X, Y)$ denotes the largest generalized eigenvalue of $\alpha Y - X$ with $Y > 0$, i.e., the largest eigenvalue of the matrix $Y^{-\frac{1}{2}} X Y^{-\frac{1}{2}}$. In our case, Eqn. 5.59, Q and $\sum_{i=0}^{d} p_{ij} \widehat{A}_i^T Q_i \widehat{A}_i$ are equal to x and $A(x)$, respectively.

On the other hand, if we fix the Q_i's, it is a eigenvalue problem to minimize the maximum eigenvalue of a matrix that depends affinely on a variable, subject to an LMI constraint (or determine that the constraint is infeasible)

$$\begin{aligned} \text{minimize} \quad & \alpha \\ \text{subject to} \quad & \alpha I - \sum_{i=0}^{d} p_{ij} \widehat{A}_i^T Q_i \widehat{A}_i Q_j^{-1} > 0 \end{aligned} \tag{5.64}$$

where $\alpha I - \sum_{i=0}^{d} p_{ij} \widehat{A}_i^T Q_i \widehat{A}_i Q_j^{-1}$ are symmetric matrices that depend affinely on the optimization variable K, the controller gain matrix. This is a convex optimization problem. Both of these problems can be solved very efficiently by MATLAB's LMI toolbox or SeDuMi[28] through convex optimization [2].

Note that we want to design a controller for a specified transition probability matrix P. However, it is often difficult to obtain an initial guess that works well for this P, So we start with a simple P_0 for which it is easy to design a stabilizing

controller, and then change P in an outer loop as we proceed through the design iteration.

1. Design a LQR (or LQG for dynamic output feedback) controller K for the plant, Eqn. 5.41, without considering delays in the loop. Let $K_i = K$, for $i = 0, 1, ..., d$, and initialize the transition probability matrix, let $P = P_0$.

2. Design Iterations

   Repeat{

   (a) V-step. Given the controllers K_i, $i = 0, 1, ..., d$, solve the LMI feasibility problem, Eqn. 5.60, for all $j = 0, ..., d$ with $\alpha = 1$ to find Q_i, $i = 0, ..., d$ to prove that the K_i's stabilize the jump system.

   (b) K-step. Given Q_i, $i = 0, ..., d$ found in V-step, solve the eigenvalue problem, Eqn. 5.61, to find the K_i, $i = 0, ..., d$ which maximize the decay rate of the closed loop jump system with respect to the Q_i's.

   (c) $\triangle$-step. Perturb the transition probability matrix P by adding a small perturbation matrix $\triangle$: $P = P + \triangle$.

   } Until the desired transition probability matrix P is reached or the V-step is not feasible.

Since the LQR (LQG) controller corresponds to the no delay case, a reasonable initial transition probability matrix is

$$P_0 = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & \dots & 0 \end{bmatrix} \simeq \begin{bmatrix} 1-(d-1)\epsilon & \epsilon & \dots & \epsilon \\ 1-(d-1)\epsilon & \epsilon & \dots & \epsilon \\ \vdots & \vdots & \vdots & \vdots \\ 1-(d-1)\epsilon & \epsilon & \dots & \epsilon \end{bmatrix} \tag{5.65}$$

This particular matrix has zero in all the rows except the first row that represents no delay. However, zero entries will cause some computation problem because Eqn. 5.59 contains inverse of probability matrix' entries in its elements. To solve it numerically, we replace p_{ij} by a very small positive number ϵ whenever it is zero in our structured transition probability matrix, Eqn. 5.46. At the same time, we must slightly change the matrix elements in the same row to keep the constraints being satisfied. For ϵ small enough, the first V-step will always be feasible. For each succeeding iteration step we add a small perturbation matrix $\triangle$ to the transition probability matrix. However, each entry of $\triangle$ must be small enough to make the next V-step have a feasible solution. This routine will repeat itself until the desired transition probability matrix P is reached or the V-step becomes infeasible. Therefore, the transition probability matrix P that is used in the last iteration may not be equal to the desired one and there is no guarantee that the desired transition probability matrix is achievable numerically. Furthermore, from our experience, smaller perturbation matrix can always improve the difference between the resulting and desired transition probability matrix. Obviously, this will increase the number of iteration.

According to our experience, MATLAB's LMI toolbox work faster than SeDuMi when solving lower number of variables. On the other hand, SeDuMi will work faster than LMI toolbox when solving larger number of variables. For V-step, we will have $Q_1, ..., Q_d$ as the matrix variable where $d = d_a \times d_s$ is the dimension of transition probability matrix and each matrix variable $Q_i = Q_i^T \in \mathbb{R}^{q \times q}$, where $q = n(d_s+1)+m(d_a-1)$, will have $\frac{q}{2}(1+q)$ number of variable. Hence, the total number of variable need to be solve in v-step will be huge. On the other hand, there will only be $n^2 + m(2n + r)$ number of variable in k-step that is independent to the number of step delay. Hence, this number of variable needed to be solve in k-step will be minimum. As a result, we will use SeDuMi and LMI toolbox in v-step and k-step,

respectively.

5.5.5 Example: Magnetically Levitated Ball (MLB)

Consider the magnetic levitation system problem in Fig. 5.13. This is a second order unstable system. The state variables are $[y \quad \dot{y}]^T$. The parameters used are: $g = 9.81$, and $k = 0.01$. The sampling time is $T = 0.05$ second. Assume the case where random delays exist from the sensor to controller and from the controller to the actuator, and they are bounded by 2: $r(k) \in \{0, 1, 2\}$. The controllers are designed using the linearized model, which linearized at $y = 0.5$, but the computer simulation uses the nonlinear dynamics. The initial condition for simulation is $y = 1.3$ and $\dot{y} = 0.5$.

Given the "expected" transition probability matrix

$$P = \begin{bmatrix} 0.1410 & 0.8590 & 0 \\ 0.1410 & 0.4295 & 0.4295 \\ 0.1410 & 0.4295 & 0.4295 \end{bmatrix} \tag{5.66}$$

which has a stationary distribution given as follow: [0.14 0.49 0.37]. We started from an LQG controller

$$\begin{aligned} z(k+1) &= \begin{bmatrix} 0.3270 & 0.0372 \\ -5.4479 & 0.4983 \end{bmatrix} z(k) + \begin{bmatrix} 0.6368 \\ 3.9892 \end{bmatrix} y(k) \\ v(k) &= \begin{bmatrix} 54.2303 & 8.6564 \end{bmatrix} z(k) \end{aligned} \tag{5.67}$$

The initial transition probability matrix and the small perturbation matrix used are

$$P_0 = \begin{bmatrix} 0.98 & 0.01 & 0.01 \\ 0.98 & 0.01 & 0.01 \\ 0.98 & 0.01 & 0.01 \end{bmatrix}, \qquad \Delta = \begin{bmatrix} -0.0050 & 0.0050 & 0 \\ -0.0050 & 0.0025 & 0.0025 \\ -0.0050 & 0.0025 & 0.0025 \end{bmatrix} \tag{5.68}$$

The iterative design procedure terminate after 119 iterations (approximately 10 hours on a Pentium III workstation) because of an infeasible solution for V-step and the resulting transition probability matrix is $P_{118} = \left[\begin{smallmatrix} 0.390 & 0.600 & 0.01 \\ 0.390 & 0.305 & 0.305 \\ 0.390 & 0.305 & 0.305 \end{smallmatrix}\right]$ with stationary distribution $[0.39 \quad 0.42 \quad 0.19]$. There is a huge difference between the resulting and desired transition probability matrix.

One of the reason for v-step becomes infeasible is the perturbation matrix is too big.. Therefore we try another design iteration with smaller perturbation matrix

$$P = \begin{bmatrix} 0.98 & 0.01 & 0.01 \\ 0.98 & 0.01 & 0.01 \\ 0.98 & 0.01 & 0.01 \end{bmatrix}, \qquad \Delta = \begin{bmatrix} -0.0010 & 0.0010 & 0 \\ -0.0010 & 0.0005 & 0.0005 \\ -0.0010 & 0.0005 & 0.0005 \end{bmatrix} \tag{5.69}$$

The iterative design procedure provides a controller which makes the closed-loop system mean square stable for the desired P.

$$\begin{aligned} z(k+1) &= \begin{bmatrix} 0.8013 & 0.1474 \\ -2.3919 & 0.5287 \end{bmatrix} z(k) + \begin{bmatrix} 0.6535 \\ 1.7800 \end{bmatrix} y(k) \\ v(k) &= \begin{bmatrix} 1.8702 & 13.5617 \end{bmatrix} z(k) + 65.7471 y(k) \end{aligned} \tag{5.70}$$

In this case, there were 840 design iterations, but it took approximately 36 hours on a Pentium 3 workstation. Fig. 5.15 shows one simulation run of the Markovian jump delays according to the given transition probability matrix, and the initial condition response of the closed-loop system using this controller. The first part of this figure corresponds to the sensor delay sequence. For example, at 9 second, the delay steps is equal to 2 that means the controller is using sensor information provided at 8.9 second because 2-steps is equal to 0.1 second. The second part of the figure corresponds to the control delay sequence. For example, at 16 second, the delay steps is equal to 0 that means the actuator is using non-delayed controller information. Finally, the

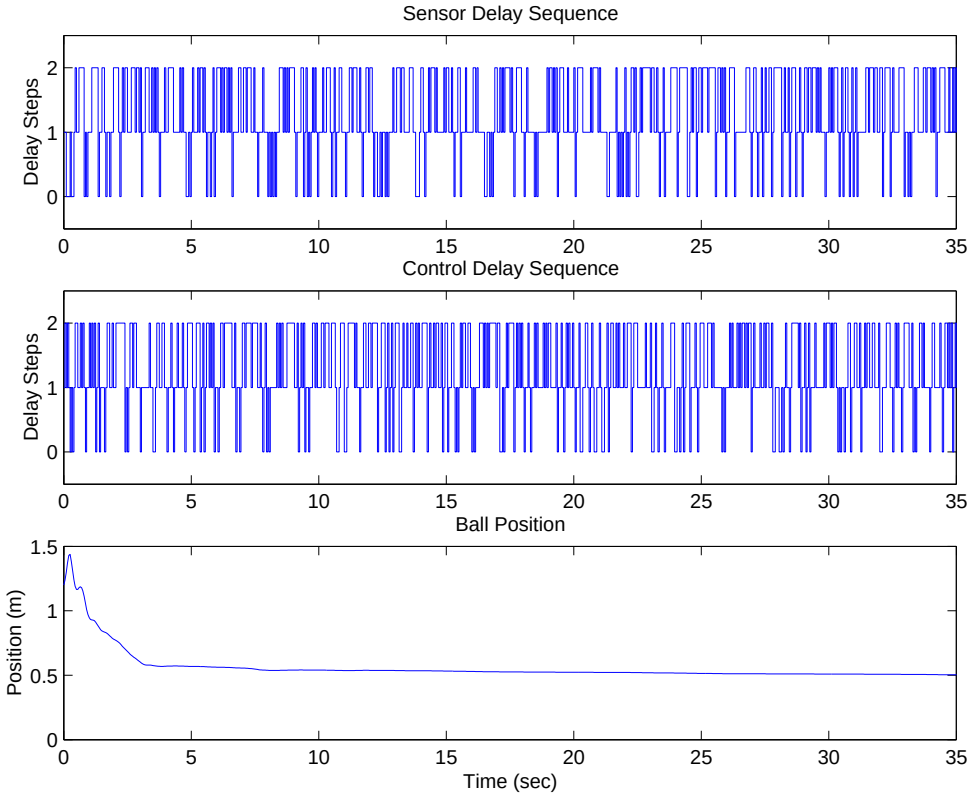


Figure 5.15: Random Delays and Initial Condition Response

last plot corresponds to the initial condition response of the closed-loop system. The closed loop Markovian jump system with the above controller has a spectral radius of 0.9955. According to Theorem 5, the system is mean square stable and Fig. 5.15 has confirmed this result. The time that is required for the system to reach steady state in Fig. 5.15 is 35 sec. This result is expected because the controller has to take a little longer to stabilize the system because of delayed information and action.

5.6 Limitation and Summary

In section 5.1, we have introduced the networked control system model, major assumption and variables that are used through the chapter. In section 5.2, we have

stated the necessary and sufficient conditions for constant delay system to be controllable and observable.

In section 5.3, we have introduced discrete resetting Smith predictor to control system with constant delay. The design process is rather simple and easy to implement. This predictor relies on complete observation for the system internal states. In addition, this controller requires huge memory storage that is directly proportion to the number of delay steps and internal states. Furthermore, this type of controller cannot guarantee stability for delay system with jump (random) behavior.

The random delay system can be modeled as a Markovian jump system. The notion of mean square stability for random delay systems is introduced in section 5.4. One interesting point is that stability of every mode is neither necessary nor sufficient for mean square stability of jump system.

In section 5.5, we model a random delay system as a Markovian jump system. By using Theorem 5.4, we can design a dynamic controller that can stabilize a system with transition probability matrix P as long as the closed loop system satisfies Eqn. 5.60. This controller does not require full observation of the system's internal states. The memory storage is directly proportional to the number of internal states but irrelevant to the number of delay steps. Furthermore, this controller can also stabilize system with constant delay. For example, for system with 1-step delay, the transition probability is equal to $P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$. However, the main disadvantage for this controller is it requires huge computation power and the design time is extremely long.

In this chapter, we have introduced controller design for networked control system. The discrete resetting Smith predictor is suitable for system with constant delay (e.g. CAN and Profibus) and full state observation. This controller is rather easy to design. However, for system that is implemented through Ethernet requiring a controller that can handle random delay, the controller design procedure is relatively long and

requires huge computation power. The main advantage is that the controller does not require huge memory storage and full state observation. We believe these two controllers cover most of the cases for linear system controller design implemented through a communication network.

Chapter 6

Conclusion

In this thesis, we have studied extensively networked control systems (NCS). The main reason for such systems to gain popularity in the control industry is because they can reduce wiring as compared to typical point-to-point wiring scheme. Furthermore, it becomes much easier for an engineer to monitor the signal flow. However, the major obstacle for NCS to be successfully implemented is the communication delay induced by the network. It is well known that delays can destabilize control systems. Therefore, this issue plays an important role in networked control systems. However, the characteristic of network delay depends on the type of network being used for systems. Therefore, we have focused our attention on three types of network: Controller Area Network (CAN), Process Field Bus (Profibus), and Ethernet.

In chapter 2, we described the protocol and delay model of CAN. CAN system can prioritize messages and determine which messages have a higher priority and should be transmitted first. It has the best error detection and recovery mechanism (with HD[1]=6) as compared to other networks. CAN can linked up to 110 nodes on a single network. It offers high-speed communication rate up to 1 Mbits/sec with maximum data size of 8 bytes. Therefore, this network is only suitable for small system with

[1]Hamming Distance

minimum data exchange. For CAN, detailed timing analysis has been studied in [12, 38]. Their models are incomplete because they did not characterize transmission error appropriately. Since CAN is a prioritized network, any retransmission on high priority messages will substantially increase the response time for lower priority messages. By including transmission errors in our model, we are able to produce a more realistic estimate of the response time. The is crucial for controller design.

In chapter 3, we described the protocol and delay model of Profibus. Profibus is a token passing network that contains master and slave stations. Master devices (active stations) determine the data communication on the bus. A master can send message without an external request when it holds the bus access rights (the token). Slave devices (passive station) are peripherals such as I/O devices, valves, drives and measuring transducers. PROFIBUS can link up to 124 stations with a maximum of 244 bytes input and output data for each slave with high-speed communication rate up to 12Mbits/sec. Therefore, this network is suitable for small system with medium data size for exchange. A detailed timing analysis has been studied in [22, 40]. Their models do not include error retransmission. It is important to include transmission error to the timing model because it is often unavoidable and will substantially increase the response time. By including transmission error to our model, we again are able to produce a more realistic estimate of the response time.

In chapter 4, we described the protocol of Ethernet. However, this protocol was designed with the needs of Office/Enterprise users in mind. Issues of paramount concern in control networks, such as real-time performance and redundancy were not part of the original designers' concerns. In section 4.2.1, we presented a traditional network configuration and showed that this configuration is inapplicable to control systems. It is because a single intensive user can affect the entire network stems. Therefore, we separate an office network into two subnetworks: "control network" and "office

network". For control network, we modeled the network as a fully switched network. A switch is placed in a Ethernet network with a dedicated segment for every node. These segments connect to a switch, which supports multiple dedicated segments (sometimes in the hundreds). This setting allows many conversations to occur simultaneously on a switched network. Ethernet offers high-speed communication rate up to 1Gbits/sec with maximum data size of 1500 bytes. Therefore, this network is suitable for large system with huge amounts of data exchange. However, Ethernet does not provide any error recovery process that will increase the possibility of retransmission, as any detected error will lead to retransmission.

In our timing analysis, we have include the "PAUSE" operation that is built into switch. The PAUSE function is specifically designed to prevent switches (or end stations) from unnecessarily discarding frames due to input buffer overflow under short-term transient overload conditions. This operation is a very simple stop-start form of flow control that can avoid discarding frames even when the short-term load increases above the level anticipated by the design. Our timing analysis is the first analysis that consider the "PAUSE" operation.

In chapter 5, we discussed controller design for networked control systems. In section 5.3, we discussed discrete resetting Smith predictor [21, 20] for systems with constant delay. This predictor is easy to design and implement. However, it requires full observation of systems' internal states. Furthermore, it requires large memory storage for long delay. Therefore, this predictor is only suitable for system with short and constant delay. However, for systems which are implemented through Ethernet may need a controller that can handle random delay because Ethernet does not have any error recovery process. Therefore, any detected error will lead to retransmission resulting in random delays. In section 5.5, we described a controller design procedure for system with delay that can be modeled as a Markovian jump system. This

controller design procedure is relatively long and require huge computation power. However, it does not require large memory storage and full state observation.

The networks studied in this thesis, CAN, Profibus, and Ethernet, cover small and large systems, and systems that require small or large amount of data transfer. Furthermore, we have provided controller designs suitable for systems implemented through these communication networks. We believe that we have provided guidance to system engineers planning to design system through communication networks.

6.1 Future Work

In terms of delay models, we can take some statistical measurement of packet loss from the control network. By using these measurement, we can improve our delay model. Furthermore, for Ethernet, we can consider the case where multiple LAN switches are used in the control network. In terms of controller design, the amount of time require to solve the Markovian jump system is enormous. It is because there is a large number of equations and variables to solve. The existing software (e.g SeDuMi[28] and Matlab's LMI toolbox) is not efficient. Therefore, a more efficient algorithm is needed to solve the LMI problem. Finally, we may also consider non-Markovian delay system because it is possible for some network behavior to be dependent on the past history of the network.

Appendix A

Hamming Distance

Data together with the parity bits form codewords, which are transmitted via physical connections. If we assume we have k data bits, there are 2^k different groups of data words. However if one data word is received by the receiver having one, two, or more errors, how could the errors be detected and finally corrected? If there is a single error in the received data word, the receiver cannot detect it because it represents another correct data word. For example, If we have 3 data bits, then:

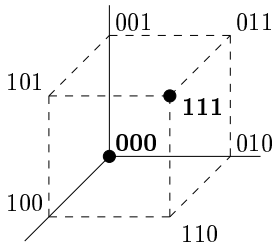
The possible form		
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

If the word 101 is transmitted and the third bit is received in error, then it would be 100. This is also one of the correct data words and there is no way that the receiver may know the original word has been changed. Therefore, additional bits must be added to the word with certain properties. First, they must be determined only

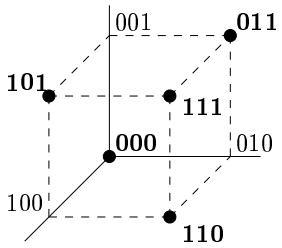
by using the current data word (memoryless). Second, they must be unique. That is, for a unique data word, a unique parity word must be generated and maintain a certain distance (bit difference) between other codewords. As mentioned before, the received codeword with error, must not only be taken as another codeword, but also it must be close in form to the original codeword. This is important for error correction. There must be a minimum bit difference between each codewords (data words plus parity words) in order to let the received error codeword be recognizable and be correctable. Therefore in the case where two data words are similar to each other (e.g., 100 and 101), the parity words must be very different (e.g., 00 and 11) to maintain the minimum bit difference in form of codeword (e.g., 10000 and 10111). Therefore, if the first codeword had an error in the fifth bit (10001), then it could be recognized that this is not a correct codeword and it is also recognized that it could have been 10000 but not 10111.

A very simple scheme is sending a single data bit with two other parity bits. The 3 bits form a 3 dimensional cube, where all possible codewords are situated at vertices. Let the one be accompanied by two other ones and zero with two other zeros. Therefore, 111 is meant for 1 and 000 is meant for 0. However, if there occurs a single error, then the receiver is able to recognize the actual word. For example, if we receive 101, and a single error has occurred, then we conclude that it must have been 111 and the second bit has been changed. However, if a second bit is also received by error, then there is no way to know which one has been sent. This scheme is being shown in Fig. A. As it is obvious, only 000 and 111 have the reliable distance of three from one another. This assures up to one bit error correction. By referring to the $2t + 1$ law we also find that for one error correction there must be at least 2+1=3 bit in any codewords.

For error detection, there is an another criterion. If we assume for one error bit,



then receiving a non-valid codeword would be counted as error detection. This is shown in Fig. A.



This scheme explains that in order to have error correction, the spheres (with unit radius) around each codewords must not intersect or touch (tangent) other spheres. In case of intersecting, all the common points bear ambiguity, such that, if a codeword is being exposed to error and fall in this common space, there is no way to identify to which original codeword it belongs. Referring the $t+1$ rule, it specifies that in order to have t error detection, the codewords must be separated by a distance of $t+1$.

Bibliography

[1] Klaus Bender, editor. *PROFIBUS: The fieldbus for Industrial Automation.* Prentice Hall, 1993.

[2] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory.* SIAM, Philadelphia, 1994.

[3] Manrique Brenes. Industrial ethernet-gaining factory floor strength. *World Bus Journal*, pages 6–9, Sep 2002.

[4] CAN CiA. Controller area network. http://www.can-cia.ru/RU-CAN02.pdf, 2001. Presentation.

[5] N-Tron (The Industrial Network Company). Why use industrial ethernet switches? http://www.n-tron.com/pdf/why_use_industrial.pdf, 2003.

[6] Oswaldo L.V. Costa and Marcelo D. Fragoso. Stability results for discrete-time linear systems with markovian jumping parameters. *Journal of Mathematical Analysis and Applications*, 179:154–178, 1993.

[7] Statnett SF Enterprise. *Norwegian IEC 870-5-101 User Conventions*, 3 2000.

[8] PROFIBUS Nutzerorganisation eV and Profibus Trade Organization. *PROFIBUS – System Description*, 10 2002.

[9] Laurent El Ghaoui and Mustapha Ait Rami. Robust state-feedback stabilization of jump linear systems via lmis. *International Journal Robust and Nonlinear Control*, 1996.

[10] S. Giordano, A. Lombardo, M. Meo, and G. Schembra. Overview of the research results on "Source Modeling". http://www1.tlc.polito.it/mqos/courmayeur/sorgenti/sor2.pdf, 1999.

[11] Alexander Graham. *Kronecker Products and Matrix Calculus with Applications.* Ellis Horwood Limited, 1981.

[12] Jong Man Jeon, Dae Won Kim, Hong Seok Kim, Yong Jo Cho, and Beom Hee Lee. An analysis of network-based control system using CAN (Controller Area Network) protocol. In *IEEE - International Conference on Robotics and Automation*, volume 4, pages 3577–3581, Seoul, Korea, May 2001.

[13] Y. Ji and H.J. Chizeck. Jump linear quadratic gaussian control: Steady-state solution and testable conditions. *Control Theory and Advanced Technology*, 6:289–319, 1990.

[14] S.A. Koubias and G.D. Papadopoulos. Modern fieldbus communication architectures for real-time industrial applications. *Computers in Industry*, 26:243–252, 1995.

[15] Wolfhard Lawrenz. *CAN system Engineering From Theory to Practical Applications.* Springer, New York, first edition, 1997.

[16] Alberto Leon-Garcia. *Communication Networks.* McGraw-Hill Companies, Inc., 2000.

[17] Feng-Li Lian, James R. Moyne, and Dawn M. Tilbur. Time-Delay Modeling and Optimal Controller Design of Networked Control Systems. Technical report, University of Michigan – Cell Automation Group, August 2001.

[18] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computer Machinery*, 20:46–61, 1973.

[19] M. Malek-Zavarei and M. Jamshidi. *Time-Delay Systems: Analysis, Optimization and Applications*. Elsevier Science Publishers, The Netherlands, 1987.

[20] S. Mondie, P. Garcia, and R. Lozano. Resetting smith predictor for the control of unstable systems with delay. *IFAC 15th Triennial World Congress*, July 2002.

[21] Sabine Mondie, Rogelio Lozano, and Joaquin Collado. Resetting process-model control for unstable systems with delay. *Proceedings of the 40th IEEE Conference on Decision and Control*, pages 2247–2252, December 2001.

[22] Salvatore Monforte, Mario Alves, Franciso Vasques, and Eduardo Tovar. Designing real-time systems based on mono-master PROFIBUS-DP networks. *16th IFAC Workshop on Distributed Computer Control Systems*, November 2000.

[23] Bill Moss. Real-time control on ethernet. *Dedicated Systems Magazine*, pages 53–60, Q2 2000.

[24] N. Navet, Y-Q. Song, and F. Simonot. Worst-case deadline failure probability in real-time applications distributed over CAN (Controller Area Network). *Journal of Systems Architecture*, 46(7):607–617, 2000.

[25] Johan Nilson. *Real-Time Control Systems with Delays*. PhD thesis, Lund Institute of Technology, 1998.

[26] Paul G. Otanez, Jonathan Parrott, James R. Moyne, and Dawn M. Tilbury. Analysis of distributed control networks for reconfigurable machining systems. *ERC/ITIA Workshop on Control of Manufacturing Systems*, February 2002.

[27] Z.J. Palmor and Y. Halev. Robustness properties of sampled-data systems with dead time compensators. *Automatica*, 26:637–640, 1990.

[28] Dimitri Peaucelle and Kristen Taitz. http://www.laas.fr/∽peaucell/SeDuMiInt.html, 2002.

[29] Olaf Pfeiffer. Making medical devices smarter with can and canopen protocols. http://www.devicelink.com/mem/archive/02/10/pfeiffer.html, 2002.

[30] Hans-Christian Reuss. Extended frame format - a new option of the CAN protocol. Technical Report HAI/AN 92 002, Product Concept and Application Laboratory Hamburg, F.R. Germany, 5 1993.

[31] Mondie S., Dambrine M., and Santos O. Approximation of control laws with distributed delays: a necessary condition for stability. *IFAC Conference on Systems, Structure and Control*, pages 541–551, August 2001.

[32] Rich Seifert. *The Switch Book: The complete guide to LAN switching technology.* John Wiley & Sone, 2000.

[33] O.J.M. Smith. Closer control of loops with dead time. *Chemical Engineering Science*, 53:217–219, 1957.

[34] IEEE Computer Society. *IEEE standard 802.3*, 3 2002.

[35] Charles E. Spurgeon. *Ethernet: The Definite Guide.* O'Reilly and Accociates, Inc., 2000.

[36] Peter Szemes, Zoltan Puklus, and Karoly Biro. Time delay compensation for internet based control. *International Workshop on Trends and Recent Achievements in Information Technology*, pages 122–127, May 2002.

[37] K. Tindell and A. Burns. Guaranteed message latencies for distributed safety-critical hard real-time control networks. Technical report, University of York (UK), 1994.

[38] K. Tindell and A. Burns. Guaranteeing message latencies on Control Area Network. *Proceeding 1st International CAN conference*, September 1994.

[39] K. Tindell, H. Hansson, and A. Wellings. Analysing real-time communications: Controller Area Network. *Proceeding 15th Real-Time Systems Symp.*, December 1994.

[40] Eduardo Tovar and Francisco Vasques. Analysis of the worst-case real token rotation time in PROFIBUS networks. *Proceedings of the 3rd International Conference on Fieldbus Systems and their Applications*, September 1999.

[41] Eduardo Tovar and Francisco Vasques. Real-time fieldbus communications using PROFIBUS networks. *IEEE Transactions on Industrial Electronics*, 46:1241–1251, 1999.

[42] J. Unruh, H.-J. Mathony, and K.-H. Kaiser. Error detection analysis of automotive communication protocols. Technical report, Robert Bosch GmbH, 1989.

[43] Josef Weigmann and Gerhard Kilian. *Decentralization with PROFIBUS-DP*. MCD Verlag, 2000.

[44] J.D. Wheelis. Process control communications: Token Bus, CSMA/CD, or Token Ring? *ISA Trans.*, 32:193–198, 1993.

[45] Lin Xiao, Arash Hassibi, and Jonathan How. Control with random communication delays via a discrete-time jump system approach. *Proceedings of the American Control Conference*, pages 2199–2204, June 2000.

[46] H. Zeltwanger. Controller area network: A serial bus system - not just for vehicles. http://www.esd-electronics.com/german/PDF-file/ CAN/Englisch/intro-e.pdf, 2000.

[47] Wei Zhang, Michael S. Branicky, and Stephen M. Philips. Stability of networked control systems. *IEEE Control Systems Magazine*, pages 84–98, 2 2001.