Conceptualization of Temporal Specifications for Application to Distributed Real-Time Control Systems

This Thesis is presented for a degree of 
Doctor of Philosophy (PhD) in Engineering

By 
Rachna Dhand

M.C.A (Masters in Computer Applications), 2000

School of Engineering and Energy 
Murdoch University, Murdoch, Western Australia 
2011
Dedicated to my son
Arnav Seth
&
my husband
Amit Seth
# Contents

List of figures................................................................................................................... iv
List of Tables...................................................................................................................... vi
Declaration........................................................................................................................ vii
Abstract.............................................................................................................................. viii
Acknowledgement........................................................................................................... ix
Publications........................................................................................................................ x

## Chapter 1 Introduction

1.1 Industrial Control Systems: An Introduction......................................................... 1
1.2 Categories of Industrial Automation Systems......................................................... 5
1.3 Role of Delays in Control System Operation........................................................... 6
1.4 Research Objective................................................................................................. 9
1.5 Thesis Outline......................................................................................................... 9
1.6 Tools, Techniques & Packages Used...................................................................... 11
1.7 Thesis Plan and Information Flowchart ................................................................. 11
1.8 References............................................................................................................. 12

## Chapter 2 Research Overview and Literature Survey

2.1 Problem Definition ................................................................................................. 13
  2.1.1 Real-time Languages: Brief Overview................................................................. 14
  2.1.2 Literature Overview......................................................................................... 17
2.2 Problem Definition................................................................................................. 22
2.3 Research Methodology........................................................................................... 23
2.4 References............................................................................................................. 25

## Chapter 3 Control Loop Implementation using Real-Time (RT)-Languages

3.1 RT-Languages: A Foreword.................................................................................... 27
  3.1.1 Physical Computation Model.............................................................................. 28
  3.1.2 Bounded Computation Model............................................................................ 29
  3.1.3 Zero Computation Model.................................................................................. 32
  3.1.4 Logical Computation Model............................................................................. 33
3.2 Control Software Engineering Methodology.......................................................... 38
3.3 Temporal Specification and Mathematical Modelling for Time Delayed System.... 41
3.4 Pragmatic Modelling of Control Algorithm using RT-Languages....................... 47
  3.4.1 Using TDLVisualCreator and functionality code in Java................................ 48
  3.4.2 Using MATLAB/SIMULINK............................................................................. 50
3.5 Conclusion.............................................................................................................. 54
3.6 References............................................................................................................. 54
Chapter 4  Communication Delays: Modelling and Impact on Real-Time Control System 56
4.1 Communication Delays 56
  4.1.1 Characterization and Modelling 56
  4.1.2 Modelling Variations for Networked Control System 58
4.2 Auto-Regressive Integrated Moving Average Method (ARIMA) 60
  4.2.1 An Introduction 60
  4.2.2 Prediction and Diagnostic Checks 66
4.3 Experimental Set-up for Communication Delay Modelling 68
  4.3.1 Protocol Analysis: DataSocket Transport Protocol (DSTP) and PROFIBUS Decentralised Periphery (DP) 68
  4.3.2 Experimental Set-up for Ethernet and PROFIBUS DP 72
  4.3.3 Ethernet Traffic Characterization, Modelling and Prediction with ARIMA 74
  4.3.4 PROFIBUS DP Traffic Analysis and Modelling 85
4.4 Conclusion 87
4.5 References 88

Chapter 5  Programmatic Rationale for Delay Compensation And Controller Design 89
5.1 Introduction 89
  5.1.1 Mathematical Modelling for Networked Time-Delayed System 90
    5.1.1.1 Delays Smaller than Sampling Period 91
    5.1.1.2 Delays Larger than Sampling Period 93
  5.1.2 Delay Step Estimation Algorithm 94
5.2 Model-Based and Adaptive Controller Design 96
  5.2.1 Model-Based Control 96
  5.2.2 Dahlin Control Algorithm 99
  5.2.3 Adaptive Control 100
    5.2.3.1 Introduction 100
    5.2.3.2 Methodology 102
  5.2.4 Metrics for Closed-Loop Response Analysis 107
5.3 Delay Compensation Algorithms for Deterministic Networks 108
5.4 Delay Compensation Algorithms for Non-Deterministic Networks 115
  5.4.1 Comparative Analysis for Smith Predictor, Dahlin and Self-Tuned Regulator for Random Delay Compensation 125
5.5 Conclusion 126
5.6 References 127
Chapter 6  Conclusion  129
  6.1 Conclusion  129
  6.2 Future Research  133
  6.3 Publications  133

Appendices  135
Appendix ‘A’: Time Definition Language (TDL) &
  TDLVisualCreator: A Tutorial  136
  A.1 Time Definition Language  136
  A.2 E-Machine and TDL Compiler  140
  A.3 References  141

Appendix ‘B’: DataSocket Transport Protocol (DSTP): An Overview  142
  B.1 Data Transfer API Designed in Java  143

Appendix ‘C’: Inter-Arrival Gaps for 1 Workstation over Ethernet/DSTP  152
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Communication Architecture for a Distributed Control System</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>Discrete or Sampled Control-Loop Operation</td>
<td>7</td>
</tr>
<tr>
<td>1.3</td>
<td>Research Plan Chart</td>
<td>11</td>
</tr>
<tr>
<td>2.1</td>
<td>Research Methodology Flowchart</td>
<td>24</td>
</tr>
<tr>
<td>3.1</td>
<td>Control System Design Methodology</td>
<td>40</td>
</tr>
<tr>
<td>3.2</td>
<td>Two noninteracting water tanks</td>
<td>42</td>
</tr>
<tr>
<td>3.3</td>
<td>Computation Time for Non-Delayed System using TDL</td>
<td>50</td>
</tr>
<tr>
<td>3.4</td>
<td>TDL/SIMULINK Model for Water-Tank System</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>a) Controller TDL Module Expanded using TDL Library</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>b) SIMULINK Model with TDL Module as Controller</td>
<td>51</td>
</tr>
<tr>
<td>4.1</td>
<td>PROFIBUS/OSI Network Model</td>
<td>70</td>
</tr>
<tr>
<td>4.2</td>
<td>Control Loop with Communication Links between Sensors and Actuators</td>
<td>72</td>
</tr>
<tr>
<td>4.3</td>
<td>PROFIBUS Experimental Set-up</td>
<td>74</td>
</tr>
<tr>
<td>4.4a</td>
<td>Control Engineering Laboratory at Murdoch University for Ethernet Traffic Modelling</td>
<td>75</td>
</tr>
<tr>
<td>4.4b</td>
<td>Traffic Traces for Ethernet over a period of 30 Seconds</td>
<td>76</td>
</tr>
<tr>
<td>4.5</td>
<td>Cumulative Distribution Function for Publication Time for One to Four Workstations</td>
<td>77</td>
</tr>
<tr>
<td>4.6</td>
<td>Hurst Estimates for Ethernet Traffic Samples</td>
<td>80</td>
</tr>
<tr>
<td>4.7</td>
<td>Random Walk Models for One to Four Workstations</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>a) Autocorrelation and Partial Autocorrelation for One Workstation</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>Response Time Gaps</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>b) Autocorrelation and Partial Autocorrelation for Four Workstation</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>Response Time Gaps</td>
<td>82</td>
</tr>
<tr>
<td>4.8</td>
<td>Residual and Forecast Analysis</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>a) Residuals and Fits for One Workstation Response Time Gaps</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>b) Residuals and Fits for Four workstation Response Time Gaps</td>
<td>84</td>
</tr>
<tr>
<td>4.9</td>
<td>PROFIBUS DP Traffic Trace for a Period of 30 Minutes</td>
<td>86</td>
</tr>
<tr>
<td>5.1</td>
<td>Control Signal Generation</td>
<td>91</td>
</tr>
<tr>
<td>5.2</td>
<td>Smith Predictor</td>
<td>96</td>
</tr>
<tr>
<td>5.3</td>
<td>Block Specification for Explicit Self-Tuned Controller</td>
<td>103</td>
</tr>
<tr>
<td>5.4</td>
<td>Control Loop for Self-Tuned Bányász and Keviczky Adaptive Algorithm</td>
<td>107</td>
</tr>
<tr>
<td>5.5</td>
<td>Closed-Loop Response Metric Analysis</td>
<td>108</td>
</tr>
<tr>
<td>5.6a</td>
<td>Computation of Controller Module in 30 ms</td>
<td>111</td>
</tr>
<tr>
<td>5.6b</td>
<td>Smith Predictor with Direct synthesis Control</td>
<td>111</td>
</tr>
<tr>
<td>5.7</td>
<td>Computation of Controller Module in 30ms</td>
<td>113</td>
</tr>
<tr>
<td>5.8</td>
<td>Dahlin Control</td>
<td>113</td>
</tr>
<tr>
<td>5.9</td>
<td>Smith Predictor for Varying Delay Magnitude within Control-Loop</td>
<td>117</td>
</tr>
<tr>
<td></td>
<td>a) SIMULINK Model for Control-Loop Operation with Ethernet/DSTP Communication Link</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>b) Smith Predictor Subsystem Expanded</td>
<td>118</td>
</tr>
</tbody>
</table>
Figure 5.10  Closed-Loop Response for Smith Predictor
Figure 5.11  Dahlin Algorithm for Varying Delay Magnitude
  within Control-Loop
  a)  Dahlin Control for Delay Compensation
  b)  TDL Module Expanded
Figure 5.12  Closed-Loop Response for Dahlin Controller
Figure 5.13  Adaptive Self-Tuned Regulator
  a)  Adaptive Self-Tuned Control block in TDL/SIMULINK
  b)  ARX Plant Estimation TDL Module Expanded
Figure 5.14  Adaptive Self-Tuned Regulator for Random Communication Delays
  a)  Controller Output
  b)  Process Output with Adaptive Controller
Figure A.1  TDL Library in SIMULINK
Figure A.2  TDL VisualCreator
Figure A.3  TDL/SIMULINK Model
Figure A.4  Task Definition for Module
Figure A.5  TDL Code Viewer and Period Specification
Figure A.6  TDL Task Execution Environment
Figure A.7  TDL Mode Structure
Figure A.8  Task Time Cycle
Figure B.1  GUI Interface for DSTP Connection
List Of Tables

Table 3.1  Factors Differentiating Programming Abstractions at Performance Level  35
Table 4.1  Descriptive Statistics for Inter-Arrival Gaps  77
Table 4.2  Average Publication Time Delays (in Milliseconds)  78
Table 4.3  Fitted SARIMA Model Details for One and Four Workstation Response Time Gaps  83
Table 5.1  Response Time estimate and Computation Time in TDL/SIMULINK  114
Table 5.2  Response Time Estimates  126
Appendix ‘C’: Inter-Arrival Gaps for 1 Workstation over DSTP/Ethernet  152
Declaration

I declare that this thesis is my own account of research and contains, as its main content, work which has not been submitted before for a degree at any tertiary educational institution.

---------------------------------

Rachna Dhand
Abstract

The main consideration in the design of any control system is the intensive use of sensor equipment and its integration with a real-time computing environment. The degree of integration determines the efficacy and complexity of the control artefact. This research aims at investigating the use of Real-Time Languages as a generic paradigm for designing distributed automation systems. A challenging problem in control of any networked industrial plant is minimization and variance of delay within its control-loop. The time delay in executing the control algorithm originates from:

1. The computation giving rise to the control decision;
2. The sampling time chosen if a discrete-time controller is used;
3. Communication delays due to network characteristics like the application protocol in use, the topology, or the type of physical communication hardware used;
4. Physical transport delays as an integral part of plant operation.

A distributed real-time control system, linked through a communication network, is bound to be affected by the randomness of communication delay patterns. The research focuses on optimizing the controller to account for communication and computation delays. The computational delay can be bounded by the application of Real Time (RT)-languages such as the Timing Definition Language (TDL). Real-time programming methodologies like RT-Java or TDL have evolved in the last decade to address the increasing complexity of control systems and scalability of equipment in the industrial automation domain. Different temporal execution models are analysed and their execution is explored using RT-languages.

Communication networks like Ethernet have unbounded delays. Statistical modelling techniques, like Auto-regressive Integrated Moving Average (ARIMA), may be used to model the network traffic. The case study for Ethernet traffic modelling serves as a benchmark for modelling communication delays that are random in nature. Comprehensive coverage is provided for network traffic modelling through the stochastic approach ARIMA, with a case study of National Instruments (NI) DataSocket Transport Protocol (DSTP), based on high bandwidth Ethernet.
In real-time control systems, the controller optimization requires accurate temporal specification of sensitive controller tasks. Logical computation languages such as TDL have successfully eliminated the temporal unpredictability of designing control software. Finally, this thesis constructs an analytical and programmatic rationale on the impact and compensation of unpredictable network delays through discrete-time control algorithms, that are designed in TDL. An adaptive self-tuned regulator is designed for compensating variably bounded delays. The results validate that discrete implementations of the Self-Tuned Controller are able to compensate for delay, thus guaranteeing the stability of the control-loop in the presence of random delays.
Acknowledgements

I would like to thank my supervisor, Dr. Gareth Lee, for his support and valuable guidance during the course of this research. I am extremely honoured to have him as my potential supervisor and mentor. I am thankful to my co-supervisor, Dr. Graeme Cole for his valuable support during the experimentation course of my research.

I am extremely obliged and thankful to Dr. Parisa Bahri, Dean, School of Engineering and Energy, for her guidance, support and suggestions. Her extreme support and concern led my way towards the timely completion of this project.

Most importantly, I am thankful to Dr. Wolfgang Pree, CEO and Founder, PreeTEC and Dr. Andreas Naderlinger, Researcher, PreeTEC, for donating the TDL Tools specifically E-Machine. Their donation has been extremely important to this project.

I would like express my sincerest gratitude to Foad Samadi, PhD Student, Electrical Engineering- Simon Fraser University, British Columbia, Karne de Boer, PhD Student, Murdoch University and Mehdi Moghaddam, PhD Student, Murdoch University, for their valuable suggestions in designing the real-time codes for this project. Sincere thanks to staff and PhD Students at the School of Engineering and Energy. I would like to thank particularly to Roselina Stone and Will Stirling for their administrative support and help.

I am extremely delighted to express my sincere gratitude towards my family, my husband Amit Seth, and son Arnav Seth, for their constant encouragement and unconditional support. I am thankful to my parents and sister for their care and love shown during this period. Finally my sincere acknowledgement to Murdoch University for funding the research with Murdoch International Scholarship and HDR staff for their timely support.
Publications

Journal Publications


Under Review


Conference Publications


1.1 Industrial Automation Systems: An Introduction

Industrial control systems encompass the domains of management, manufacturing, production planning, strategic planning, process control and production control. Software for process control has been developing rapidly and attempts are being made to standardize the profusion of terminology, languages, and software packages. The prime objective of a distributed automation system is acquisition of data and plant control. The conventional and characteristic stages namely pilot study, planning of hardware implementation, application design, hardware installation, and control of plant and its optimization are well-known from the development and realization of control systems [1]. As stated in [1] “The importance of appropriate design methods for distributed real-time systems is, however still not completely understood and has scope to be improved”. The key issue in control system architecture is intensive use of control equipment and its integration with a real-time computing environment. The integration level determines the efficacy and complexity of any control system.
This project deals with applicability of modern tool set and methodologies for designing distributed control systems. The modern set of tools and methodologies encompass Real Time (RT)-Languages like Real Time-Java and Time Definition Language (TDL); Remote Procedure Call Packages like RT-Publish/Subscribe and Statistical Tools for Control Network Traffic Modelling. The stability of a control system depends on the individual components that make up the control system architecture. These components are: the plant, the controllers, sensors for data acquisition and actuators. These components are linked together through a shared communication medium often referred to as real-time or control network. A typical distributed control system (DCS) architecture for the industrial processes uses switched data communication lines for network and data acquisition. The commonly used communication channels are a combination of Switched Ethernet and Foundation Fieldbus. Other standards such as PROFIBUS, Controller Area Network (CAN), or PROFINET are widely used. The use of varied communication networks and instrumentation equipment makes data acquisition issues through general communication software more complex.

An industrial control system working on the top of a real-time communication network is referred to as Distributed Control System. The major functional areas encountered in many control applications are as follows: data acquisition and conditioning, digital control, supervisory control and sequence control [1]. Data Acquisition and Conditioning packages perform basic steps of scanning, filtering, conversion and limit checking [2]. Generally, distributed control systems make use of Supervisory Control packages like SCADA for control data management and data acquisition. These packages are meant for computing set-point values or changes for analog or direct
digital controllers in continuous processes [2]. The computations are specified using either standard adjustment equations or using special simplified procedural languages [2]. Direct digital control packages are primarily based on the digital implementation and subsequent tuning of conventional three-term PID Control Algorithm. The sub-algorithms like PI, P or I can also be used depending on the requirement. Sequence control packages serve for programming start-up/shut-down operations or batch processes in a continuous manner. The Figure 1.1 depicts the Distributed Control System (DCS) architecture for an arbitrary Industrial Plant.

The traditional distributed control system architecture involves extensive use of supervisory packages for control and data acquisition. These packages utilise proprietary protocols, being been designed by industries to suit their system communication requirements. For instance, SCAN 3000 is a Honeywell product that performs data acquisition, conversion, display, supervisory and sequence control. In 2003, Honeywell launched the Experion Process Knowledge System (PKS) that integrates a knowledge based system with process modelling and unlike the SCAN 3000 system is capable of decision making. It has platform support for varied control networks including Foundation Fieldbus, HART, PROFIBUS, CAN, and Control Net. Experion and SCAN 3000 rely on proprietary protocols developed specifically for process automation.

Apart from SCAN 3000 and Experion, OLE for Process Control (OPC) is a Distributed Component Object Model (DCOM) based Microsoft standard for process automation systems [3]. OPC is good a example of how OLE-based middleware is used in the
control domain [3]. OPC standardizes the communication of process control data. It provides interoperability between the diverse plant hardware components. It has provided the standard specifications for data acquisition through diverse plant hardware components like PLCs, sensors or actuators.

![Communication Architecture for a Distributed Control System](image)

**Figure 1.1 Communication Architecture for a Distributed Control System**
1.2 Categories for Industrial Automation Systems

Not all industrial plants need to operate in real-time. Depending on the time requirements for control loop operation within a plant, industrial plants can be categorized as:

1. Hard Real-Time Control Systems
2. Soft Real-Time Control Systems
3. Hybrid Real-Time Control Systems

Hard Real-Time Control Systems are industrial control systems, that have response time specified as an absolute value that cannot be violated. Even very low probability of time violation can lead to destabilization of the system, hence causing disastrous impact. Therefore such systems are also referred to as safety critical systems. These systems have fixed time constraints and are sensitive to even single violation in time specification. Examples of such systems are aircraft flight control, anti-lock brake systems within cars and jet engine control systems. The real-time constraint within the anti-lock brake system is a short time constant ($\tau$), to prevent wheels from locking.

Soft Real-Time Control Systems are industrial control systems, that can tolerate low probability of time constant violation. These systems specify response time as an relative value and there is room for occasional violation of the time constraint. Frequent violations can lead to destabilization. Examples are process control systems that display longer time constants, several processes related to paper industry, distillation columns, and catalytic reactors.
Hybrid Real-time Control Systems are industrial control systems, that specify both hard and soft time constraints for the processes. The class of processes specified with strict time constraints have to achieve the relative performance within that specified time. Examples of such systems are industrial manufacturing, robotic control systems, and plant operations in petrochemical industries. For instance, in industrial manufacturing safety interlocks or alarms may be hard but process control is soft (due to longer time constant ($\tau$)).

1.3 Role of Delays in Control System Operation

The control action for a sophisticated industrial plant can be modelled as a three step process:

a) data acquisition from sensors;

b) control law processing and appropriate control action determination by controllers; and

c) control action sent to actuators.

This three step process forms a single control loop operation within an industrial plant. An industrial plant may have hundreds of such control loops. The aim of this research is to model various sources of delay within a control loop of a distributed control system, of a form that might apply to hard real-time systems e.g. avionics or model predictive controllers. The control loop operation, when implemented through a digital computer, introduces a digitized view of the process in which information is discretized over time. The continuous signals are discretized over a fixed sampling period.
Therefore computer controlled discretized systems are often referred to as Discrete-Time Control Systems. The discrete control loop operation is summarized in Figure 1.2.

\[ e(k) = y(k) - y_m(k) \]

**Figure 1.2 Discrete or Sampled Control Loop Operation**

The discrete-time systems need to sample the continuous signals, filter and possibly multiplex the signals and finally reconstruct them over discrete-time intervals. The signal conditioning and reconstruction introduces digital filters and zero-order holds (ZOH) in the control loop operation, as shown in Figure 1.2. Signal reconstruction is required to give a discrete signal the semblance of being continuous. The digitized control loop operation gives a new notion for time delays within a system. Now the various sources of delay within a system are:

a) the plant itself (i.e. transport delays);

b) the control software;

c) the control network (i.e. communication delays);

d) electronic delays in sensors and actuators;

e) delays associated with filtering or conditioning.
Correspondingly, the time delays can be categorized as:

i. Computation Delays that originate from the plant operation and control action simulation.

ii. Network or Communication Delays that originate from the network medium.

The computation delays are stochastic and frequently of the order of milliseconds. Consequently, the systems specifically categorized as soft or hybrid real-time systems, do not get affected by the small magnitude and randomness of the computation delays. The network delays are not bounded and have greater variance. A system which is digitized over a very small sampling period may get affected with the greater variability of network delay, because with a shorter sampling period the network load will increase. This, in turn, can lead to congestion thus causing delay or loss of information over the network. The network delays are often stochastic and depend on many factors like the network protocol in use, routing mechanisms, topology, physical transport like wired or wireless, and bandwidth characteristics of the system.

Both types of delays have negligible affect on systems with high time constants. But the delays can destabilize systems where time constants are very short. For instance, in hard real-time systems, where the magnitude of time constant for control loop operation is very short, even the slightest delay can degrade the system performance. If the computation and network delay exceed the sample time, the loss or delay problem can destabilize the system operation.
1.4 Research Objective

My research analyses the time specification for the individual components within control loop operation, using appropriate software tools like real-time (RT) languages and statistical approaches.

The basic aim is to apply modern set of tools and methodologies for implementing the distributed control system operation. The modern set of tools and methodologies encompass RT-Languages like RT-Java and Timing Definition Language (TDL); Statistical tools such as ARIMA Modeling of control network traffic delay estimation.

The brief outline of research aim is to:

a) **Provide the abstract notion of service-based computing for automating the time sensitive tasks in the control system software engineering using TDL.**

b) **Secondly, to model and analyze the impact of network-induced delays by industrial automation network protocols.**

c) **Finally, to compensate for the delay to provide a stabilized control behavior through predictive and adaptive control techniques, for enhanced real-time characteristics of a control-loop operation within a plant model.**

The application of this research is strictly in the domain of hard real-time systems where the time constants of the plant operation are very short.

1.5 Thesis Outline

My thesis is organized in six chapters that outline the details of research objectives specified in the previous section. This chapter presents a brief overview of research topic with specification of research objectives that will be achieved in later chapters.
Chapter 2 provides a more detailed literature survey and research overview.

Chapter 3 provides the programmatic rationale for control law design in RT-Java and TDL for a real-time control systems.

Chapter 4 provides an overview of network delay modelling for deterministic and non-deterministic networks. The case studies utilized for delay modelling are DataSocket Transport Protocol for High Speed Ethernet and PROFIBUS DP. The delay characteristics are examined and delay pattern is finally modelled using ARIMA for Ethernet traffic and average delay for PROFIBUS DP.

Chapter 5 provides an integration framework for variable network delay and optimized control performance. Finally the effects of network delays on control performance are analysed, compared using Simulink and compensated using a Smith Predictor, an Adaptive Dahlin algorithm and an Adaptive self-tuned algorithm.

Chapter 6 provides the conclusion of the thesis with detail of publications regarding the work done.
1.6 Tools, Techniques & Packages Used

RT-Languages: Java/Time Definition Language (TDL), Real-Time Java, PreeTEC’s TDL Visual Creator

Packages: MATLAB/Simulink, Minitab 15, EasyFit 5.1 Professional

Stochastic Techniques: ARIMA with Seasonal Component for Ethernet/DSTP Traffic

1.7 Thesis Plan & Information Flowchart

Figure 1.3 Research Plan Chart
1.8 References


Chapter 2  Research Overview & Literature Survey

Key Points

.....Problem Definition

.....Literature Survey

.....Research Methodology

2.1 Problem Introduction

My research examines methods to minimize the effects of random delays from the network in a control loop of a distributed control system. A methodology was developed using Java combined with the Timing Definition Language (TDL) Real-time Language. The plant model for optimized delay-tolerant control relates to two different aspects of control system design. One domain relates to the computational advancements made in the form of real-time languages like TDL [1], Esterel, or Hierarchical Time Language [2]. The development of real-time languages and scheduling policies like Earliest Deadline First (EDF) or Rate Monotonic (RT) have made the control system design predictable, reliable and accurate [6].

Another domain relates to analysis of networked control systems, the nature of delays and finally their compensation using optimized control laws in closed-loop operation. My research focuses on three main aspects:
1) Designing a practical model using the predictable real-time language TDL, for stable control loop operations.

2) Analysis of network communication delays using stochastic techniques (such as ARIMA) and creation of predictive models of delay behaviour.

3) Finally compensation for random delays within the control loop is manifested through:
   a) A Time-stamped Self-Tuned Adaptive Controller in case of delays that change randomly;
   b) A Time-stamped Digital Smith Predictor for bounded delays.

2.1.1 Real Time Languages: Brief Overview

Real-time programming methodologies have evolved over several decades to meet the increasing complexity of digital control design. The programming paradigms have evolved over time from Physical Computation and Scheduled Model [3] to high level time-based abstractions, such as Zero Computation [4] and Logical Computation models [5]. The details of these computation models are discussed in Chapter 3.

The scheduled or bounded computation model enhances the operating system support for real-time programming by introducing real-time scheduling policies like Earliest Deadline First (EDF) [6] and Rate Monotonic (RM) [6]. Most real-time operating systems, Real-Time Java and Ada were developed to use scheduled or bounded computation time models. The scheduled model aims to achieve concurrent execution of real-time tasks with bounded deadlines. Each task must complete its execution prior to
The most recent programming abstraction is the Logical Computation Time (LCT) Model. LCT assumes that the task has to complete within a logical time that excludes the reading and writing time of the control data. Giotto, Timing Definition Language (TDL) and Hierarchical Time Language (HTL) all follow the logical computation time model. Both ZCT and LCT models were developed for application to digital control design. Esterel is a dataflow language and lacks an interaction framework for use with high level languages such as Java and C++ whereas Timing Definition Language (TDL) is high level Java compiler-based time language.

The LCT model assumes that execution time of the concurrent control tasks complies with logical execution time (LET) [5]. LET excludes the I/O time of the tasks. The input and output time of the control task is independent of its logical time. The task has to
complete its execution within the logical time. As explained by Henzinger and Kirsch, “LCT produces code composed of multiple tasks with their I/O times specified to ensure concurrent execution is restricted only to the extent necessary to be consistent with I/O dependencies between the tasks. This enables model-based design in the sense that the model is directly translated into semantically equivalent executable programs” [10, 11]. Therefore, languages developed using these programming abstractions are closely linked to digital control design frameworks, like Simulink. TDL has manifold advantages as a pragmatic modelling tool for control system design such as:

1) TDL is a time specification language;
2) Logic is still encoded in high-level languages such as Java or C++;
3) TDL produces a timed code that acts as a wrapper around the task generated by the high level language compiler;
4) TDL guarantees the execution of a control task within the specified period called Logical Execution Time (LET) under the assumption that WCETs are known and hold;
5) TDL has a well-defined interface with common digital control simulation packages like Simulink.

2.1.2 Literature Overview

Adaptive strategies for control law optimization using least square criteria were first introduced by Kalman [10]. Later the adaptive algorithms were modified and designed using pole assignment [11] in the Laplace domain. Predictive control algorithms with single and multi-step predictions were subsequently introduced [12] [13]. Digital PID controllers were developed, which used three parameter (controller gain, integral and
derivative time constants) algorithmic estimation techniques [14]. Explicit self-tuning PID Controllers were introduced by Corripio and Tomkins using algorithmic adaptation [15]. The explicit self-tuned controllers were originally designed to minimize system output variance using the general minimization of output dispersion method developed by Clarke and Gawthrop [15] [16]. Later, much attention was paid to multi-step prediction adaptive algorithms [17]. Digital PID controllers which might be able to use parameter estimates gained through recursive identification of a plant were surveyed by Bobal et. al [16]. One of the most popular algorithms is the Dahlin digital PID Controller that calculates the controller output using an incremental relation, discretized by the forward rectangular method [17, 18].

Feedback control systems wherein the control loops are closed by digital communication networks are referred to as networked control systems [23, 24]. Investigating the stability of distributed control system has been a particular focus for researchers over the past decade. The impact of communication delays on control loop performance was analysed by Ray [19] and Branicky [20]. They considered continuous-time plant and discrete-time controllers and analysed the impact of communication links. They assume a clock-driven controller and the system is represented by a state vector made up of past values of plant input and output [20]. Wittenmark et al. [21], have studied this stochastic problem using the state space model, and finally developed stability criteria and used Linear Quadratic Controllers. In pioneering work, Nilsson [22] studied the control domain as discrete-time systems, and categorised network delays as either constant, independently random or governed by Markov chains. He devised the Stochastic Optimal Controller with time stamping, used later in this research
[25], for compensating delays with constant magnitude. He devised a sub-optimal controller based on the standard LQG controller and assumed that the network delays are governed by Markov chains to accommodate randomness [23]. None of the above mentioned researchers provided any experimental validation of their results.

From the literature survey, it is apparent that the stability of networked control system may be dealt with two different ways:

1) Branicky and Wittenmark [21, 22], analysed delayed control system architectures and devised optimized control algorithms to compensate for known network delays.

2) Walsh and Bushnell [24], analysed and devised scheduling algorithms for networks. They assumed a bounded delay originating from communication links leading to so-called deterministic networks. The notion of Maximum Allowable Transfer Interval (MATI) was introduced by Walsh et al. [24]. Their work aimed to optimize the sampling time to compensate for the delay arising from communication links within the control loop. The two scheduling policies: try-once-discard (TOD) [24] and token ring type static scheduling [24] were devised. For each of these protocols, an upper bound of MATI can be computed that ensures at least one transmission within the sampling time. This, in turn, preserves the stability of control loop operation in the presence of communication delays.

Generally, networks can be classified as:

a) Deterministic Networks: These are the real-time networks which follow scheduled access for packet transmission. The Medium Access Control (MAC) protocol for networks like FDDI [25], Token Ring (IEEE 802.6), Industrial Automation
Networks like PROFIBUS and Foundation Fieldbus with Token Bus architectures fall under this category. Such networks often use a token passing mechanism rather like the schedule of a radio station that ensures each program gains access to the airwaves within a 24 hour period. The nodes can communicate on the network according to the pre-determined and repetitive pattern that ensures that every node gets access within a fixed period. Therefore these networks are referred to as Time Deterministic networks. The periodic transmission of messages is managed through scheduling algorithms, e.g. the Time Triggered Protocol in PROFIBUS [26]. The communication delays among the nodes result from just the waiting period for the token. This waiting period can be bounded. The domain of scheduled access within communication links only applies to deterministic networks like Fiber Distributed Data Interface (FDDI), PROFIBUS or Token Ring. Walsh et. al [24] devised two scheduling policies: try-once-discard (TOD) [24] and token ring. Token ring uses static scheduling [24] for bounding the delay within deterministic networks whereas TOD uses dynamic scheduling for allocating the network resources.

b) Non-Deterministic Networks: These are the networks in which the MAC protocol randomly accesses the network for the packet transmission across the network. Ethernet [27] and Controller Area Network (CAN) [28] are non-deterministic networks. Ethernet uses the carrier sense multiple access with collision detection (CSMA/CD) algorithm. With this access method, the station will transfer the data packets only if the medium is idle. Once the transmission starts, if it detects a collision, the nodes back off from transmission for a random period of time until the corrupted data packet is retransmitted. Consequently the communication delays in
such networks are random and cannot be bounded. Such networks are referred to as non-deterministic networks. Controller Area Network (CAN) is a control network that uses a priority based CSMA protocol [28].

The directions of the work discussed in the previous paragraphs have led into two different domains: control law optimization and real-time communication link scheduling. The convergence and experimental analysis of their work was neither tested nor discussed. Controller validation was achieved without assuming predictable modelling for network delays. The random communication delays were assumed to be derived from Markov modulated chains. This assumption is too specific for mathematical modelling of delays that originate from varied networks. Walsh et. al [25] have tried to model and specify the precise mathematical behaviour of delays originating from deterministic networks.

The major drawback that exists with the previous work, regarding stability of control-loop operation, is that it is in the form of mathematical theories, that were not tested experimentally. The applicability of real-time languages such as Timing Definition Language (TDL) or Esterel for automating the control loop operation was not analysed by any of the researchers discussed previously. Therefore the advance of real-time community regarding the time-bound execution of control tasks was totally ignored. No experimental validation has been provided for the optimal controllers designed to compensate for communication delays. It is important to check the validity of controllers on a experimental set-up for its practical relevance. None of the above researchers laid stress on optimum mathematical or stochastic modelling of the delays.
The common assumption is that delay follows a certain probability density function, which is insufficient to know the exact nature of delays within the network system that operates randomly. A probability distribution function (PDF) or a Cumulative Distribution function (CDF) can provide the information about the most likely outcome of delay over time. For instance, the most likely outcome of a biased dice may be predicted by a PDF. For data originating from a random source, the conditional probability models are more suitable. In order to effectively design a control loop in the presence of communication delay, it is important to have a stochastic model that can accurately emulate and predict the delay originating from a network. The notion of maximum allowable transfer interval introduced by Walsh et. al is bound to deterministic networks. For non-deterministic networks, control strategies need to be designed that compensate for communication delay through controller optimization.

Previous work on stability of a networked control system can be grouped under a priori analysis of networked control systems. The policies and scheduling frameworks have their domain limited to deterministic networks. Moreover the delay is always considered to be bounded, with low levels of burstiness and high autocorrelation, which is not true for non-deterministic networks like Ethernet. In this scenario, the previous work done is inappropriate for networked control systems using Ethernet or CAN.
2.2 Problem Definition

The research described here follows a *posteriori analysis* which states that a novel way to stabilize the control system architecture, with the communication delays originating from the non-deterministic networks, is to:

1) estimate the delay magnitude by appropriate models such as ARIMA and generate a precise stochastic model for the delay behaviour;
2) optimize the controller a-priori for delay compensation.

Delays are categorized as originating from a either non-deterministic or a deterministic network, or computation delays originating from the digital computers that execute the control law. Consequently, controller optimization can be achieved, both considering communication and computation delays. The computation delay can be bounded by the application of RT-Languages like TDL. The obvious advantage of programmatic plug-ins designed in TDL is the time-bounded computation of control tasks. The Dahlin algorithm, Smith Predictor [29] and Adaptive Self-Tuned controller will be used to compensate for delays originating from deterministic and non-deterministic networks and compared for their performance in further chapters. Finally communication delays are bounded using a control loop designed in TDL and:

1) Time-Stamped Digital Smith Predictor [30] [31] and Adaptive Dahlin Algorithm for the delays that originate from deterministic networks
2.3 Research Methodology

The research methodology employs a posteriori analysis and control loop implementation through digital algorithms. A three-step procedure was used for incremental optimization of the control loop considering both computation and communication delays. The methodology is summarized in Figure 2.1
Temporal Specification Issues For Real-Time Control Systems

Analysis Phase

- Computation Delays
- Communication Delays

Deterministic Networks
Non-Deterministic Networks

Stochastic and Pragmatic Modeling of Delays

System Modelling

- TDL/Java, RT-Java
- Stochastic Network Delay Predictor Case Studies

- Ethernet: ARIMA Modelling
- PROFIBUS: Probabilistic Modelling

Delay Optimization using a posteriori analysis

System Design

- Adaptive Digital Self-Tuned Dahlin algorithm & Discrete Smith Predictor for Deterministic Networks
- Adaptive Dahlin and Self-Tuned Bányász & Keviczky Predictor for Non-Deterministic Delays

Figure. 2.1 Research Methodology Flowchart
2.4 References


Real-time programming methodologies have evolved in the last decade to address the increasing complexity of control systems and scalability of equipment in the industrial automation domain [1]. As explained by Kirsch, and Sengupta, the programming abstractions for real-time systems can be classified according to the processor computation time cycle for a given control task [2]. The programming paradigms have evolved from conventional physical and bounded computation time [3] paradigms to modern programming models that incorporate the notion of logical execution time [4].

Most real-time control systems are designed using conventional languages and empirical methodologies. Control software engineering methodologies need to take advantage of modern programming paradigms in order to have guaranteed time specifications. Temporal specifications are to be met, not for enhanced performance, but for timing precision in real-time computing. Time specifications for a real-time control
system can be hard or soft. Safety-critical systems often require hard temporal constraints that cannot be violated. Programming languages for real-time computing have evolved from low-level to high-level and fall into the following three classes based on time specification:

a) Physical Computation Time (PCT) Model
b) Bounded Computation Time (BCT) Model [2]
c) Logical Execution Time (LCT) Model [4]
d) Zero Computation Time (ZCT) Model [5]

Each case will be dealt with separately in the subsections that follow.

3.1.1 Physical Computation Model

The physical computation time (PCT) model assumes the time specification for control task execution to be constant. The tasks are usually programmed in a low-level language such as assembly language. The control tasks are deterministic. The tasks are coded to generate the list of instructions. The clock cycles are determined for each instruction and summed. Finally, the total computation time is determined from the clock speed. Since the task has exclusive access to the processor, total clock cycles are proportional to the elapsed time thus guaranteeing the execution of control task within time specification. However this is not true if the code handles interrupts or it is threaded or the operating system is multitasking. This is the way that much microcontroller software is still written for embedded devices such as MP3 players or electronic devices. The PCT programming results in precise real-time behaviour and enables high I/O throughput [2]. The major drawback with this programming abstraction is that it lacks the ability to handle concurrent tasks. The real-time programming models developed
after the PCT model were designed with compositional aspect to guarantee the real-time constraints of multiple concurrent tasks.

3.1.2 Bounded Computation Time Model

The bounded computation time model assumes that the process has to complete its execution within a worst case execution time (WCET). The execution time is constrained by real-time bounds for each control task [2]. Bounded computation programming is also referred to as the “Scheduled model” [2]. The process is said to be schedulable in bounded computation time if and only if it is able to complete its execution within WCET. The process can be pre-empted from the execution cycle and diverted onto another task. The WCET is specified at compile time and cannot be changed during execution. Most real-time operating systems follow the scheduled model execution algorithms like Rate-Monotonic [6] and Earliest Deadline First [6].

The Rate-Monotonic algorithm is used for scheduling periodic tasks. The priorities are proportional to the activation frequency of the tasks. These priorities are assigned at the time of task creation and cannot change. Therefore, the Rate-Monotonic algorithm is also referred to as the static scheduling algorithm, whereas, Earliest Deadline First is a dynamic priority scheduling algorithm. The task priorities are based on the absolute deadlines of the control tasks. The task with earliest deadline receives the highest priority. Since the absolute deadline changes in the case of a multithreaded control task, the priority also changes.
Many real-time languages such as Ada and the Real-Time Java (RT-Java) are based on the Scheduled Model [2]. For example, Real-Time Java (RT-Java) follows the scheduled model semantics. Java was not initially designed to comply with real-time semantics. Two early initiatives brought into Java were:

a) Real-Time Java Specification [7] by Sun Microsystems and IBM did not intend to change the Java semantics in order to maintain its backward compatibility, instead added new features to the JVM and standard packages for classes.

b) Real-Time Core by J Consortium [8] introduced a new API for real-time tasks that was based on PERC Systems. It was unable to gain acceptance and was finally dropped.

Real-Time Java enhanced Java in the following areas:

a) Packages for classes abstracting real-time threads: The `RealtimeThread` is a Schedulable object that implements the `Schedulable` interface. `RealtimeThread` extends the basic `Thread` class within Java and redefines a few methods to meet its requirements. For instance, the `Interrupt` method within the class is redefined to implement asynchronous transfer of control, a new `Sleep` method is defined to interact with `RealtimeClock and Time` classes and a `currentThread` method is defined to identify the currently executing thread as the real-time thread.

b) Schedulable Resource management using priority-based scheduling of real-time threads: Standard Java allows each thread to have a priority that can be used by the JVM. JVM relies on the underlying operating system support. Therefore, it is not essential that the highest priority thread is executing at any time instant. Real-Time Java introduced the notion of schedulable objects [2]. Each schedulable object implements a `Schedulable` interface. This interface is
implemented by two classes: *RealtimeThread* and *AsynEventHandler*. The dispatcher within the scheduler supports the pre-emptive priority-based dispatching of the schedulable objects.

c) Classes providing real-time clocks and time specification through standard interface definitions: The *HighResolutionTime* [2] class enables time values to be expressed with nanosecond precision. This class is an abstract class and has three subclasses. The subclasses are named as the *RelativeTime* Class, the *AbsoluteTime* Class and the *RationalTime* Class [2]. The relative time and absolute time duration can be measured with respect to the user time source. Rational time is relative time which measures the rate at which a particular event occurs.

d) Standard interfaces for asynchronous transfer of control to handle sporadic and aperiodic control tasks: Real-Time Java handles asynchronous events as dataless occurrences that are triggered by interrupts [2]. For efficiency of event-based control, real-time asynchronous events and handlers are defined. A single event can be associated with one or more than one handlers or a single handler can be associated with multiple events. Each handler stores the pending event occurrences in a variable referred to as firecount. The firecount is incremented and the associated handler is released when the event occurs.

Real-Time languages based on the BCT model are well suited to use the real-time scheduling policies of operating systems. Control systems are designed using the principles of linearity [18]. Correspondingly, the control software engineering needs to follow the rules of composition [8] for all the components of control system. As stated
in [8], "Compositional rules state that a complex multi-component system can be
designed from its constituent components". As stated in [8] [9], the compositional
framework was developed as a two-step process:

a) “Component Abstraction: The real-time behavior of parent component is
derived in a real-time interface” [8] [9]. Thus real-time interface is a collective
abstraction of real-time properties of parent component.

b) “Component Composition: The real-time interface is synchronized for the
common events for multiple components thus preserving the principle of
compositionality” [8] [9]. The local timing properties represented through real-
time interfaces are composed into a global timing property [9].

The Scheduled model is unable to accommodate changes that occur in the I/O behaviour
of control tasks at run-time. The scheduling decisions for the concurrent control tasks
are predefined and based on their WCETs. If a new process enters the system, the
revised scheduling decisions are hard to redefine using the BCT programming
abstraction. Once a BCT system has been released, it is hard to change it again, because
of system-wide scheduling effects [2]. The scheduled model is compositional towards
the computation pattern of control tasks but not towards the I/O behaviour.

3.1.3 Zero Computation Model

It is important to analyse I/O behavior of the real-time processes along with their
computation pattern. The requirement for I/O determinism was induced by Synchronous
languages assume that the tasks are input determined [2], which states that for all
sequences of input values and times, the program produces unique sequences of output
values and times [5]. Control modelling tools like Simulink have synchronous reactive
semantics. The semantics follow the rules of deterministic programming for the I/O part
of an application – the part that interacts with inputs, monitors and controls the signals and events in environment [12]. It assumes that control tasks are executed immediately and in zero computation time. Hence this semantic is also referred to as zero-time computation modelling [10]. Suppose the system is in a given state. When an event occurs, the synchronous semantics causes the system to react and enter into a new state. The time computation for this change is assumed to be zero. Esterel is a specification language which follows this semantic. Esterel has restricted integration with other languages. The compiler produces sequential code which is executed by the underlying operating system. The tasks are input-determined and are generated as sequential instructions with no support for real-time scheduling policies like Earliest Deadline First [6] or Rate-Monotonic [6]. The disadvantage is that the model does not take advantage of the real time capabilities of newly developed RT-operating systems. It generates accurate time specification for tasks that are input-determined, but determinism at computation level is not guaranteed.

3.1.4 Logical Computation Model

The most recent programming abstraction is Logical Computation Time Model (LCT) [13, 14]. The programming model was introduced by Giotto [14], a real-time programming language. The LCT model strictly assumes the notion of logical time or worst case execution time, but it excludes the time for I/O operations, such as reading and writing, unlike the BCT model. Thus logical time for a control task is the time taken to complete its computation tasks. The input and output time of the control task is independent of its logical time. The task has to complete its execution within the logical time and is available to write output, only after the logical computation time has
elapsed. Thus the LCT model provides compositionality both towards the input behaviour and the execution pattern for the tasks. The model incorporates run-time support from RT-Operating systems, thus providing a superset of the features of BCT model. The code written in real-time language supporting LCT model is portable and has predictable computation time. Timing Definition Language (TDL) is a real-time language that supports the LCT model and is a successor of Giotto.

TDL is a timing language. The time safety concept introduced by this language can make the LCT model perform better for hard real-time systems. TDL integrates well with simulation and modelling environments such as Simulink [15]. TDL has a well-defined interface referred to as TDLVisual Creator with Simulink. The TDL VisualCreator tool is discussed in detail in Appendix ‘A’. TDL executes the control algorithm in two steps, thereby separating the platform dependant issues like schedulability, from platform independent issues like generating code from a given Simulink model of the system. The two-step execution process followed by TDL is:

1. **Program Generation** [14] from a given mathematical model that is described using MathWorks Simulink. The mathematical model is transformed to a timing module that wraps the control tasks.

2. **Compilation** [14] of the timing code into an executable for the platform. This step will ensure the schedulability is achieved for the system specific to the execution environment.

Table 1 summarizes the features of the execution models for Real-Time Languages and compares them.
Table 1 Factors Differentiating the Programming Models at Performance Level

<table>
<thead>
<tr>
<th>Factors</th>
<th>Models</th>
<th>BCT Model</th>
<th>Synchronous Reactive</th>
<th>LCT Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language Support</td>
<td>RT-Java, Ada, Erlang</td>
<td>SIMULINK, Esterel, Lustre</td>
<td></td>
<td>Giotto, Time Definition Language (TDL)</td>
</tr>
<tr>
<td>Platform Independence</td>
<td>Yes</td>
<td></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Language Semantics</td>
<td>Imperative &amp; Functional</td>
<td>Data Flow</td>
<td>Time Triggered</td>
<td></td>
</tr>
<tr>
<td>Real-Time Scheduling Policies</td>
<td>Required</td>
<td></td>
<td>No</td>
<td>Only Run-Time Support</td>
</tr>
<tr>
<td>EDF &amp; Rate Monotonic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task Determinism</td>
<td>Execution Time</td>
<td>Compile Time</td>
<td>Both</td>
<td></td>
</tr>
<tr>
<td>Achieved at</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>API for RT-Support</td>
<td>Abstract Classes and Packages</td>
<td>Sequential Code Generation Execution in Zero-Time Fully State Based Model</td>
<td>Time Triggered Architecture</td>
<td></td>
</tr>
<tr>
<td>MATLAB/Simulink Interactional Framework</td>
<td>Absent</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Deadline = Worst Case Execution Time (WCET). It includes the I/O and computation time.
TDL wraps the functionality i.e. the control algorithm, into a timing module. The functional code specifies the control tasks and the timing code guarantees real-time computation of the control tasks within their logical time.

TDL enhances the concept of component based component design in which the timing is segregated from functionality of a control task. The control tasks and their implementation details rely on high level languages such as Java. TDL guarantees the execution of functional code under the assumption that the WCET are known and are valid. The TDL module is a top-level component that fulfils the real-time specifications of a control task, independent of its functionality. TDL produces a timing module for the control tasks that is automatically generated in Java/C++ and executed by the so-called E-Machine. Details of the language semantics are discussed in Appendix ‘A’. The timing code is compiled into E-Code that is executed by the E-Machine [16]. E-Code is a low-level timing code that determines when tasks are released, read input and write output [2]. TDL supports the notion of component based programming by defining a module for each control task. Multiple tasks can be defined within a single TDL module. These multiple tasks are (logically) executed in parallel within a mode.

TDL provides a programming abstraction for specifying timing constraints. As explained in [2], TDL assumes that every real-time task has to complete within its period, and that the length of the period is equal to its LET. LET introduces fixed timing patterns where input values and output values are written – independent of the actual execution time of the controller software.
TDL introduces the concept of Logical Execution Time (LET) [2], which assumes that implementation of a periodic task is correct if it completes execution within the time limit equal to LET. LET [2] assumes that computation time of the process excludes the time taken to read the input and write the output. If a process has a period of 10ms, and it spends 8ms in computation, then LET will be 8ms. TDL is solely a specification language so task functionality can be handled through C/C++/Java compilers. TDL produces a compiled version of the timing specification, called the E-Code, which is executed by a virtual machine referred to as the Embedded Machine [15]. The job of E-Machine is to execute stubs within the functional code so that they start at the correct time and complete within the specified LET. TDL fills the semantic gap between digital control design and implementation.

A TDL program is made up of modes where each mode is made up of several concurrently executing tasks. TDL can be in one mode at a time. It allows switching between modes. A TDL switch facilitates the mode change by specifying the target mode. If any task within the target mode is not satisfying the schedulability constraint than the switching is cancelled.

All inter-task communication is done through TDL ports. A TDL program also specifies sensor or actuator ports. Every task has unique input and output ports. Data exchange is done through TDL drivers that interact with ports. A specification code produced by TDL is executed by the E-Machine. Task Scheduling and optimizations of control activities is managed by Scheduler referred to as S-Machine.
3.2 Control Software Engineering Methodology

The control software engineering in the component-based approach, yields a task-based process control system. The objective is to segregate timing and functions within control tasks and distribute them as components. This programming discipline offers benefits in the verification, optimization, integration, and reuse of embedded components. The most obvious advantage is the use of platform independent virtual machines and object-oriented design. The use of virtual machines enhances the portability of the code. The basic objective is to analyse the distributed control system:

a) Mathematically, for time delays that originate from external sources such as the communication medium and their compensation through discrete-time algorithms;

b) Statistically for communication delay forecasting;

c) Programmatically to ensure stability when experiencing computational and communication delays;

d) Finally, to provide a prototype for the system that reflects the delays from various sources and compensates them effectively.

The design methodology used in this research is adaptive development with incremental prototyping. The aim is to provide a rationale for automating the time sensitive tasks in control system using TDL and Real-Time Java. The control software engineering cycle for the development process can be summarized in Figure 3.1. The Figure depicts the design flow spiralling outwards. As is evident from Figure 3.1, the control software engineering cycle is divided into 4 phases:
a) Temporal Requirement Specification: In this phase, the time specifications are analysed for the control problem. The two time constraints analysed are computation delays and network delays.

b) MATLAB/Simulink Modelling: In this phase the Simulink models for the control problem are generated for different levels. Level 1 refers to an undelayed system whereas Level 2 considers the impact of computational delays. Level 3 deals with stochastic modelling of network delays using an Auto-regressive Integrated Moving Average Method (ARIMA) explained in Chapter 4. The control networks analysed at this level are Ethernet and PROFIBUS. Finally, level 4 deals with Simulink/TDL modelling of the time delayed system using Discrete-time algorithms specifically the Smith Predictor, Dahlin and Adaptive Tuning Controllers, explained in Chapter 5.
c) Pragmatic Modelling: In this phase the Time Definition Language (TDL) and RT-Java are used to implement the plant model for all levels. The implementation yields three prototypes referred to as P1, P2, and P3. The final prototype P3 is developed in three incremental stages. The plant model is first designed for an undelayed system, and programatically designed in TDL and RT-Java as prototype P1. The computational delays are fixed in the second stage with prototype 2 referred to as P2. The prototype P3 is the TDL implementation of a plant model that incorporates the effects of both Computational and Network Delays.
d) Validation & Checking: In this phase the Simulink models and TDL implementations are compared and validated to check the stability margins of the control system having stochastic network delays in control loop.

### 3.3 Temporal Specification and Mathematical Modelling for Time Delayed System

#### Case Study: Non-Linear Second Order System (Liquid Level in Interacting Tanks)

For an effective design of a process control system, the first step is development of an appropriate experimental and mathematical model for the system [17]. Differential equations characterize the relationships between process variables, and hence the behaviour of the system under varied states and input conditions [6]. In computer-based control systems, process variables like input state, disturbance and output specifically vary with respect to discretized time.

A simple example of a second order system is considered, that maintains the water flow in and out of two non-interacting tanks, with a strict timing constraint to reach a given water level. The aim was to analyse the control problem for real-time requirements and optimize the system to achieve these requirements.

The mathematical model for the water tank problem, as shown in Figure 3.2, [18] is a linear function with the volume of the water in the tank related as shown below:

\[
\frac{dv}{dt} = F_1 - F_{out} \tag{3.1}
\]
F₁ - Flow in (litres/min); Fₐₐₜ - Flow out (litres/min); V – Volume (litres)

Figure 3.2 Two non-interacting water tanks

The outflow of water is often modelled by the square root of the water level in the tank. Correspondingly the non-linear function for equation (3.1) can be written as:

\[ A \frac{dh}{dt} = F₁ - c \sqrt{hₐ} \]

where \( hₐ \) - height at a initial set-point level,

\( A \) - cross sectional area of the tank,

\( c \) - constant valve resistance.

Assume \( h \) and \( V \) are proportional so long as the cross-sectional area \( A \) is constant for all values of \( h \).

This non-linear model is linearized to obtain the process transfer function. Any function \( f(x) \) can be expressed in the power series specified in Equation (3.3), around the point \( x = xₜ \).

\[ f(x) = f(xₜ) + \left( \frac{df}{dx} \right)_{x = xₜ} (x - xₜ) + \left( \frac{d²f}{dx²} \right)_{x = xₜ} \frac{(x - xₜ)^²}{2!} + \ldots \ldots (\text{higher order terms up till } n) \]

(3.3a)

If second and higher order terms are ignored, then Equation 3a can be simplified to:
\[ f(x) \approx f(x_t) + f'(x_t)(x - x_t) \]  

(3.3b)

Similarly, the linearized approximation of the non-linear model is obtained by the Taylor series expansion of the nonlinear term (height) around the point \( h_0 \). The Taylor series expansion of the function in the specific case \( f(h) \), for \( h = h_0 \), is a function of a single variable height \([18]\). The approximate linear first order transfer function around a steady point \( h_0 \) can be given as:

\[
G(s) = \frac{K_2 K_1}{(\tau_1 s + 1)(\tau_2 s + 1)}
\]  

(3.4a)

where \( K_1 = (2^* h_1^{1/2})/c_1 \); \( K_2 = (2^* h_2^{1/2})/c_2 \); \( \tau_1 = 1/c_1 \); \( \tau_2 = c_1/c_2 \)

The delays that affect control-loop operation then can be categorized as belonging to:

a) Computation delays that occur due to control law processing.

b) Communication delays that occur due to network medium connecting the controller to sensors and actuators.

The Laplace Transform of a time delayed system contains the exponential term \( e^{-\alpha s} \) within the transfer function, where \( \alpha \) is the associated time delay. In the case study, \( \alpha \) is assumed to be the sum of the computation delay and network delay. Thus mathematically, the time delayed system in continuous domain is specified as

\[
G(s) = G(s) * e^{-\alpha s}
\]  

(3.4b)

where \( \alpha = \alpha_c + \alpha_n \); \( \alpha_c \) is the Computation Delay; \( \alpha_n \) is the Network Delay

So long as real-time software development methods are employed, it will always be known whether computation delays are within specified bounds. TDL guarantees control stability once the computational delays within the control loop operation are...
known. Transmission delays, on the other hand also play a vital role in any control system’s stability. The transmission delays are difficult to model and estimate because of the stochastic nature of network traffic.

The discrete-time equivalent of a continuous time-delayed system in equation (3.4b) is specified as

\[ G(z) = \frac{(b_1 z^{-1} + b_2 z^{-2})}{1 - a_1 z^{-1} + a_2 z^{-2}} z^{-D} \]  

(3.5)

where \( \alpha = D \Delta t \); \( \Delta t = \) Sampling Period; \( D \) is the Integer Delay Step;

The discrete model of the system makes critical assumptions about the controller gain and sampling time. These critical assumptions are required because it is possible that, for the same controller parameters, a stable continuous-time system can become unstable when discretized. The critical requirement for a closed-loop system is given by the characteristic equation [16]:

\[ 1 + g_p(z)g_c(z) = 0 \]  

(3.6)

where \( g_p(z) \) is the z-transformation of a continuous plant transfer function and \( g_c(z) \) is the discrete controller transfer function. For a discrete-time closed-loop system to be stable, all the roots of equation (3.6), must lie within the unit circle [18].

The discrete-time difference equation for the system derived from Equation (3.5) with delay specified as the multiple of sampling time is given as:

\[ y(k) = a_1 y(k - 1) - a_2 y(k - 2) + b_1 u(k) + b_2 u(k - D) \]  

(3.7)

where \( D = \alpha/\Delta t \); \( y \) is the process variable; \( u \) is the control variable.
The stability margins are established around $\Delta t$ for a constant delay. Smaller values of $\Delta t$ are often desirable for a discrete-time system to perfectly emulate the continuous system. However decreasing the sampling rate on the other hand can increase the computational and network load, thus leading to unpredictable delays in communication. Thus it is important to tune the sampling rate to establish a balance between network load and desired system performance [19]. For instance, the critical assumption for the sampling rate for a first-order time delayed system in the presence of a delay can be given as [16]:

$$\Delta t_{\text{critical}} = -\tau \ln\left(\frac{K K_c - 1}{K K_c + 1}\right)$$

(3.8)

where $K$ is the controller gain for the plant; $K_c$ is the controller gain for the PID; $\tau$ is the time constant for plant.

For stable control loop operation, the chosen sampling period should be less than $\Delta t_{\text{critical}}$. The actual determination of sampling rate depends on the magnitude of controller gain and time constant for the plant.

Walsh [19] provides an analytical approach to define the stability margins that apply to sampling time and network delays that originate from distributed control systems using deterministic networks. For a scalar system with sampling period $\Delta t$ and network delay $\alpha$, the allowable bound for $\alpha$ can be defined as [20]:

$$\max\left\{\frac{1}{2} \Delta t - \frac{1}{K}, 0\right\} < \alpha < \min\left\{\frac{1}{K}, \Delta t\right\}$$

(3.9)

The system is capable of tolerating a delay up of one sampling period. It may be infeasible to derive the exact stability region for general systems [19] because the
network delay varies in magnitude. The equation (3.9) can be used to estimate the bounds for sampling time for a given estimate of the network delay.

But this is applicable only for deterministic networks in which the delay is bounded. In practice, the networked control systems should be adaptive to disturbances that occur from the external environment. Defining the delay bound to be constant does not adequately describe the behaviour of many real-world control systems. Industrial control systems are made up of numerous control loops with feedback controllers. Often it is desirable to have \emph{a posteriori} analysis of the stochastic network delays and techniques to compensate for them. One approach is to model the stochastic delays and programmatically compensate the delay using discrete-time algorithms like Adaptive Self-Tuned Controller. This \emph{a posteriori analysis} involves stochastic modelling of delay through appropriate statistical techniques, forecasting the delay magnitude and compensate for it using discrete-time algorithms. The aspect of novelty in this work is that the delay is considered to be random and not assumed to be fixed or constant. Control system engineering is modified to compensate for the range of random delays using real-time language such as TDL. The Adaptive Self-Tuned Controller estimates the changing plant behaviour in the presence of the random delays and hence optimizes the controller to compensate for the delays.

The first aspect in analysing the plant or system to be controlled is to mathematically model it and define a controller algorithm. The sequence of control operations is often referred to as the control algorithm. The general non-delayed control operation sequence for the water tank problem is listed in Algorithm 3.1:
Listing 3.1

While(true)
    Read h(k)                  // Current Process Variable
    e(k) = sp – h(k)
    compute f(k) = G_c(e(k))    //Digital Delayed Control law
    write f(k)
endwhile

The delays can be modelled and randomly generated using statistical probability density functions. The case study presented in Chapter 4 uses Auto-regressive Integrated Moving Average Method (ARIMA) for Ethernet traffic modelling. Finally the forecast delay is compensated for using discrete-time and predictive techniques such as the Smith Predictor, Dahlin Algorithm and Adaptive Self-Tuning Controller. The control-loop stability is affected by communication as well as computational delays. RT-Languages like TDL have proved successful in guaranteeing the stability attributable to computational delays. The control optimization through TDL facilitates the constant computational delay for discrete-time predictive techniques. The control algorithm in Listing 3.1 can be implemented using Real-Time Java and Time definition language (TDL) to bound its computational delay. The control algorithm is finally optimized:

a) To compensate for delays that originate from external sources such as the network medium.

b) To bound the computation delay by using TDL and Real-Time Java.

3.4 Pragmatic Modelling of Control Algorithm using RT-Languages

TDL is a specification language which requires an auxiliary language (and compiler) to allow tasks to be implemented. A TDL program is a Timing Program and supervises the tasks that can be directly modelled in Simulink or designed using programming
languages like Java. The main focus is to segregate timing behaviour from task details.

The control algorithm in Listing 3.1 can be optimized for timing using TDL in two ways:

a) Using TDLVisualCreator & function in Java.

b) Using MATLAB/Simulink

3.4.1 Using TDLVisualCreator & functionality code in Java.

Listing 3.2

module Controller
{
    task ControlOperation[wcet=50ms]
    {
        uses ITAEController();
    }
    task DataSocketReader[wcet = 50ms]
    {
        uses Java_DataSocketWriter_write();
    }
    task DataSocketWriter[wcet=50ms]
    {
        uses Java_DataSocketReader_read();
    }
    start mode main[period=150ms]
    {
        task[freq=1] ControlOperation();
        task[freq=1] Java_DataSocketReader_read();
        task[freq=1] Java_DataSocketWriter_write();
    }
}

The timing specification for a control algorithm may be described as a TDL module. A TDL module defines three tasks with worst case computation time (WCET = 50ms). The tasks can be executed within their specified period for a required number of invocations i.e if a period is 150ms and worst case execution time for control task is 50ms, then the maximum task frequency for each task is 1. The task implementation can be defined in non-embedded programming languages like Java or C. The Java Listing 3.3 implements the task ITAEController() in the Java class Controller.
Listing 3.3

```java
/* UnDelayed ITAE Controller for Undelayed Process */
import com.preetec.tdl.tools.emachine.Out;
class Controller {
    static void ITAEController()
    {
        /* Data Declaration h - Process Variable; u - Manipulated Variable e - Error Signal k - loop variable*/
        e[k] = sp - h[k];  // Control Law Processing for Undelayed Process
        deltau[i] = KC * ((e[k] - e[k - 1]) + (deltat / taoi) * e[k]);
        // Control Signal Calculation
        u[k] = deltau[k] + u[k - 1];
        Out.println(u[k] + " " + time[k]);
    }
}
```

The listing 3.3 provides the overview of implementation for the task `ControlOperation()` defined by the Module `Controller` in listing 3.2. The implementation code is designed as a simple Java class that executes the control law for the water-tank example and computes the error signal within the time bound specified in the module 3.1. The control data is fetched from the DataSocket Server Manager, which is National Instrument’s control data repository software. The application protocol for DataSocket Server Manager is DataSocket Transport Protocol (DSTP). Chapter 4 presents the detailed analysis for network traffic modelling with DSTP as the application layer protocol over Ethernet. The tasks `DataSocketReader()` and `DataSocketWriter()`, in Listing 3.2, are Java Swing classes implemented to interact with DataSocket Server Manager. DataSocket Server Manager has an API defined for LABVIEW and LabWindows/CVI. In order to use the predefined API in LabWindow/CVI, the Java Native Interface (JNI) calls are invoked. The `Java_DataSocketWriter_write()` and `Java_DataSocketReader_read()` functions invoke the native LabWindows/CVI libraries for DataSocket Server Manager. JNI is used to provide the LabWindows/CVI libraries from Java. The codes designed for data transfer and interaction with DSTP are given in Appendix ‘B’.
The execution of the module specifies whether the task is completed within the WCET. If not, the TDL compiler generates a warning. The output for control law processing of above listings 3.2 and 3.3 is shown in Figure 3.2.

![Dispatch Table for module Controller](image1)

**a) WCET 50 ms and Period 100 ms**

![Dispatch Table for module Controller](image2)

**b) WCET 20 ms and Period 100 ms**

**Figure 3.3 Computation Time for Non-Delayed System using TDL**

As is evident from Figure 3.3, for the WCET less than 50 milliseconds the runtime system generates a warning (*WCET violation in Task: ControlOperation()*). Thus TDL helps in specifying the computation delays with known degree of precision. The task *ControlOperation()* is forced to complete its execution within the specified LET.

### 3.4.2 Using MATLAB/Simulink

TDL has a well-defined framework referred to as the TDL Library for Simulink. Simulink is a MathWorks’s modelling tool often used by control engineers to analyse the
behaviour of dynamic systems. It may be integrated with MATLAB and provides a GUI based environment for modelling control aspects of the Industrial Systems. For TDL, MATLAB/Simulink is a platform, where the TDL modules can be deployed and simulated. The TDL code can be designed using TDLVisualCreator tool and finally embedded in a Simulink model as a discrete block. The details of the TDL code design and Simulink/TDL Visual Creator are explained in Appendix ‘A’. The Simulink plant model for the water tank system using TDL and SIMULINK is shown in Figure 3.3.

a) Controller TDL Module Expanded using TDL Library

b) SIMULINK Model with TDL Module as Controller

**Figure 3.4 TDL/Simulink Model for Water Tank System**

As is evident from Figure 3.4a, the TDL Module comprises the definition of a sensor, an actuator and one task `DiscretePIDController()`. The functionality is handled automatically by the inbuilt MATLAB/Simulink C framework.
TDL has a variety of advantages for designing real-time control applications. These advantages are platform independence, deterministic execution of control tasks and portability. Above all, the timing aspects are designed independently of the functions, thus exploiting the familiarity and extensive features of high-level languages like Java or C. The benefit of using TDL in Simulink is that the plant behaviour can be modelled and analysed from a control engineer’s perspective.

By the way of comparison, RT-Java is a bounded computation language where the worst case execution time is the absolute time specified for reading, execution and writing events of control tasks. RT-Java has defined a real-time API in the form of classes and interfaces. The code listing 3.6 provides a simple example of real-time code in Real-Time Java.

Listing 3.6

```java
import javax.realtime.PeriodicParameters;
import javax.realtime.RealTimeThread;
import javax.realtime.RelativeTime;
public class ControllerModel extends RealTimeThread
{
    ControllerModel(PeriodicParameters ps)
    {
        super(null, ps);
    }
    public void run()
    {
        while(true)
        {
            read_sp;
            pv = getPVFromDataServer(); // Get Data from
                // Data Server
            e(k) = sp-pv;
            f(k) = ControlM(e(k)); // Controller Simulation:
                // Velocity Method as shown in Eq.1
            WritePVOnDataServer(f(k)); // Set Data to Data
                // Server
            WaitForNextPeriod();
        }
    }
    public static void main (String args[])
    {
        PeriodicParameters ps = new PeriodicParameters(new
            RelativeTime(100,0));
    }
```
Real-Time Java is a real time extension of Java that handles the scheduling of concurrent tasks effectively. Real-Time Java [10] achieves the temporal accuracy by defining an abstract class `RealTimeThread`. For a periodic task, the `RealTimeThread` class will reference parameters of type `PeriodicParameters` with relative time or duration of control task, set to a constant defined in milliseconds [10]. The constructor for `PeriodicParameters` performs the feasibility analysis for the process, for the deadline, the duration and cost enforcement [10]. If a deadline is missed, the scheduler reports the miss through the `waitForNextPeriod` method. The method returns false for a missed deadline. Real-Time Java supports the execution of periodic and aperiodic tasks through real-time threads and asynchronous event handlers.

Unlike TDL, Real-Time Java assumes that a process is schedulable only if it completes its execution within the time period equal to the worst case execution time of the process. Concurrent tasks have to complete in the time periods equivalent to their respective worst case execution times. The disadvantage of this model is that although it can effectively handle a periodic set of control processes, event triggered tasks and sporadic processes can hamper the scheduling pattern that is predefined by the processor. The task in TDL is controlled with timing constraints. Unlike TDL, Real-Time Java lacks the time-triggered approach for dealing with control processes. It provides standard API’s for timing and scheduling activities of the processes. If the
process I/O behaviour changes at run time, the scheduling procedure is unable to cope with the change and all tasks must be re-scheduled, thus making the system inflexible for re-use and more costly.

3.5 Conclusion

Scheduling and timing are fundamentally important aspects of real-time control applications. It is a major challenge to provide an effective software implementation that fits all aspects of real-time control systems. Both Real-Time Java and TDL have tried to provide standard implementations that provide timing predictability and real-time scheduling of resources. The convergence of control, automation and software engineering to service-based systems is one of the profound trends in technology today. RT-Languages like Real-Time Java and TDL can be applied to enforce critical timing specifications in such systems. The time delay in such systems is not just local to the controller tasks, but results from complex industrial automation networks. These systems thus require intelligent development techniques that model all sources of delay within the control loop.

3.6 References


Chapter 4  Communication Delays: Modelling & Impact on Real-Time Control System

Key Points

-----Network Delay Modelling & Characterization

-----Case Studies: Ethernet, PROFIBUS DP

4.1 Communication Delays

4.1.1 Characterization & Modelling of Communication Delays

Control and automation is an important research subject for the computing community because:

1) It covers a diverse set of applications in its domain, ranging from automated vehicle systems to process plants and refineries, robotics or large scale automated manufacturing units.

2) The systems can be modelled, using mathematical and computational techniques.

During the last decade, communication networks have been essential components of such complex systems. With the inclusion of communication networks, data flow within the system has become convenient. The inclusion of communication networks makes it harder to optimize with regard to time. The reason for this complexity is the data transfer delays within the communication network. Time is an important factor for any
Automation system. Almost all automation systems operate in real-time mode. Automation systems, in which response is required within a specified absolute value of time, are referred to as hard real-time control systems. For instance an automation system designed for a fighter plane has a time constant of less than a second for controlling the actuators. If sensors and actuators are connected via a communication network such as radio modems, the situation can be worsened because the network will induce a communication delay in control loop operation. Such real-time automation systems with stringent response requirements (short time constant) are often sensitive to even minor delays. Modelling and characterization of network delays is essential for predicting the performance and guaranteeing the Quality of Service for such systems.

Modelling the network behaviour for hard real-time systems is not the same as modelling it for the performance issues. Often network characterisation and modelling satisfy different subsets of the requirements. Characterisation is a purely phenomenological description of traffic, i.e. the analysis of measurement data, and the production of high-level statistical descriptions of the data [1]. Modelling, by contrast, covers the detailed low-level production of probabilistic models about the traffic that can be employed for making predictions about network behaviour [1]. In order to control the behaviour of a real-time system in the presence of delays that originate from computation and communication over a network, both characterisation and modelling are necessary. It is very important to produce a precise experimental characterisation and hence formulate the statistical or mathematical model for the delay behaviour that can be used in designing predictive and optimal controllers. The main objective behind the modelling of the network delays for a real-time control system is the generation of a
precise mathematical model to forecast behaviour over a period of time. This mathematical model can then be used in controller implementation to ensure stable behaviour of a control loop.

### 4.1.2 Modelling Variations for Networked Control Systems

Good network traffic modelling is an essential requirement for accurate estimations. The important metrics of traffic analysis are the average bandwidth, and autocorrelation that can be determined using packet count or inter-arrival time process. In hard real-time systems, the prior knowledge of network throughput is essential to model stabilized control loop operation. This prior knowledge regarding network throughput varies depending on the type of network. Networks can be categorized on the basis of their delay generation behaviour. Networks like PROFIBUS or CAN often produce a delay that can be bounded to some limiting value. Networks like Ethernet produce a random delay that is often auto-correlated and may even have long-range dependence. The condition of long range independence is that the autocorrelation function (ACF) of the time-series declines as a power-law, leading to positive correlations among widely distributed observations [2]. To have a better modelling approach for networks operating in real-time, we need to ensure certain constraints are followed. These are outlined as follows:

1) **Control Networks with bounded delay**: Networks in which the delay bounds can be specified a priori depending upon the time constant of the system. This is possible only for the control networks. Control networks like Fieldbus,
PROFIBUS and Industrial Ethernet work on token based semantics. These systems generate bounded delay over a period of time that can be compensated by adjusting the sampling time for the control system. Much research has been done in this regard and scheduling policies have been determined to adjust the maximum bound for the delay and hence the sampling time [3].

2) **Networks with variable delay patterns**: Many of the communication links in the industrial domain are Ethernet. The traffic pattern of Ethernet is affected by variable delays. Much ingenuity has been applied to designing Ethernet based networks for distributed control systems like switched Ethernet, or high speed Ethernet. Further, many industries are reluctant to shift their entire plant operations to specialised control networks because of cost and training factors. Ethernet is entirely different in behaviour from control networks and its modelling and characterisation require far more complexity to predict the data transfer patterns. Consequently, sophisticated modelling techniques are required to emulate the delay patterns of Ethernet traffic. The scheduling policies designed for networks that produce bounded delay patterns, cannot be applied for Ethernet as the upper delay bound is not fixed.

3) **Choice of Application Layer Protocol**: Often the traffic over a network is dependant on the application layer protocols that operate in the network layer of the OSI model. Some of the application layer protocols used in control networks are: DNP3 for Supervisory Control and Data Acquisition (SCADA) within industrial process control systems, MODBUS over TCP/IP, OPC (OLE for
Process Control), or DSTP for DSSM, SCADA-based networks. The application layer protocols serve as an important factor to determine the traffic behaviour of the control networks. SCADA-based systems often require active participation of application layer for data storage, maintenance and transfer. It is important to characterize the traffic of control networks based upon the application layer protocol. In order to have a detailed description of delays it is essential to use models that characterise for the traffic patterns generated by application layer protocols.

4) **Variability in traffic over time:** The models that are generated for network performance and QoS are often based upon data scattered over long periods of time may be a day or a week. The modelling domain for Real-Time control systems may not require macroscopic analysis of the data. The time constants for real-time automation systems are often small. A better approach for such systems will be collecting data over a shorter sampling intervals than over long time periods.

4.2 Auto-Regressive Integrated Moving Average Model (ARIMA)

4.2.1 An Introduction

Network traffic is prone to variable bounds of communication delays. “It has been characterized at the aggregate level with significant discoveries that packet inter-arrival times are not exponentially distributed, that there exist correlations in source-destination pairs, and that Long Range Dependence (LRD) may be the primary determinant of queuing behavior” [4-6]. In Ethernet, for instance, Carrier Sense
Multiple Access/Collision Detection (CSMA/CD) is used. When a collision occurs the traffic nodes back off and wait for a random period of time before retransmission. Therefore the traffic traces are affected by random delays. But on the other hand, traffic patterns are also governed by the application protocol in use. For instance, DataSocket Transport Protocol (DSTP) (used as a case study for traffic modelling in this thesis) uses the publish-subscribe mechanism for data transfer over the network. Whenever any subscriber registers its interest in the dataset, the entire active dataset is transferred, in the form of multiple packet transmissions for a single request. This manifests as the periodic component within the traffic patterns, that can be statistically modelled by analysing the autocorrelation patterns of the sample data. Therefore, it can be concluded that the delays within the network arrivals are governed by variety of probabilistic laws and can be modelled.

The nature of packet arrival patterns over the network is best judged through autocorrelation functions. The autocorrelation function is a mathematical tool to judge the correlation between values of the random process, distributed over time. In order to model the behaviour of a random variable over time, Box-Jenkins ARIMA analysis may be performed [7]. Autoregressive Integrated Moving Average (ARIMA) is a modelling tool for the analysis of the data distributed over time. The technique consists of extracting the predictable behaviour from the sample data and allows forecast to be made using three filters: the autoregressive, moving average, and the integration filter. Thus, an ARIMA (p,d,q) is a time series process where p is the autoregressive order, q the moving average order and d is the differencing order [7]. Many variations of ARIMA models have been applied, e.g. the seasonal ARIMA [8].
The network traffic is non-stationary in nature. Mathematically, network traffic can be modelled either as a counting process or inter-arrival process. The counting process specifies the number of packet arrivals in a time interval \([0,t]\). The inter-arrival process specifies the time interval between the two arrivals \([t_i, t_{i+1}]\). The inter-arrival process is a time difference series that can be modelled using ARIMA. The case study discussed further in this chapter uses inter-arrival time patterns for DataSocket Transport Protocol (DSTP) traffic over Ethernet. The main reason for choosing the inter-arrival time gaps for network modelling is the ease with which they can be captured. Moreover, the inter-arrival time gaps provide more insight to the nature of the traffic over DSTP protocol. The seasonality within the data was easily captured by analysing the autocorrelation patterns for the inter-arrival time gaps.

Generally the ARIMA process takes the following form [7]:

\[
W_t = \alpha_1 W_{t-1} + \ldots + \alpha_p W_{t-p} + Z_t + \ldots + \beta_q Z_{t-q} \quad \text{or} \quad (4.1a)
\]

\[
\Phi(B)(1-B)^d X_t = \theta(B) Z_t \quad \text{(4.1b)}
\]

where \(X_t\) denotes the inter-arrival process for traffic pattern; \(Z_t = X_t - X_{t-1}\); \(W_t = (1-B)^d X_t\);

\((\alpha_1, \alpha_2, \ldots, \alpha_p)\) are autoregressive constants

\((\beta_1, \beta_2, \ldots, \beta_q)\) are moving average constants

\(B\) is the backward shift operator and \(\Phi(B)\) and \(\theta(B)\) are autoregressive and moving average polynomials with order \(p\) and \(q\) respectively, and differencing order \(d\). \(Z_t\) denotes the first difference series obtained after differencing the original series. It is possible that periodic elements exists within the traffic pattern series. Often, if there
exists a periodic or seasonal component that repeats itself over fixed span, the model
can be extended to a general multiplicative seasonal model as follow [7]:
\[
\Phi_p(B)\Phi_P(B_s)W_t = \theta_q\theta_Q(B_s)Z_t
\]  
(4.2)

where \(\Phi_p\) and \(\theta_Q\) are seasonal autoregressive and moving average polynomials of the
order \(P\) and \(Q\), \(B_s\) is specified as \(B_sX_t = X_{t-s}\) is the seasonal span. The seasonal or
periodic component repeats itself at a fixed time interval. The length of the interval
determines the multiplicative seasonal span (s). The series is then seasonally differenced
using the span (s). \(W_t\) denotes the seasonally differenced series. The model in equation
(4.2) is a multiplicative seasonal ARIMA (SARIMA) model of the order
\((p,d,q)\times(P,D,Q)\) [8] [7]. The values of \(p, P, q, Q\) are determined from the
autocorrelation and partial autocorrelation functions of the series. Autocorrelation
measures the correlation between the sampled data observations at different time
offsets. Partial autocorrelation measures the excess of autocorrelation between the
sampled data observations in an autoregressive model. Given the time series \(X_t\), partial
autocorrelation for lag \(k\) is the autocorrelation between \(X_t\) and \(X_{t+k}\). Mathematically,
autocorrelation \((r_k)\) and partial autocorrelation \((\phi_{kk})\) for time lag \(k\), using the random
series \(x\) distributed over \(n\) periods of time, can be shown as in equations (4.3a) and
(4.3b) respectively [8] [7].

\[
\text{Autocorrelation } (r_k) = \frac{\sum_{i=1}^{n-k}(x_i-x) (x_{i+k}-x)}{\sum_{i=1}^{n}(x_i-x)^2} 
\]  
(4.3a)

where \(\bar{x}\) is the mean of the random series \(x\).

\[
\text{Partial autocorrelation } (\phi_{kk}) = \begin{cases} 
\frac{r_k-\sum_{i=1}^{k-1}\phi_{k-1,i}r_{k-1}}{1-\sum_{i=1}^{k-1}\phi_{k-1,i}^2} & \text{for } i = 2, \ldots, k \\
r_k & \text{for } i = 1 
\end{cases} 
\]  
(4.3b)

where \(\phi_{kl} = \begin{cases} 
\phi_{k-1,i} - \phi_{kk}\phi_{k-1,k-i} & \text{for } i = 1,2,\ldots,k-1 \\
\phi_{kk} & \text{for } i = k 
\end{cases} \)
One useful application of this model is for making predictions based on the past data values. A time series that shows trend and seasonality in its behaviour is often forecast using the Model Equation Method [7]. For a good model, the residuals are random and have small variance. For instance, given the series up to time \( n \), and the fitted residuals also up to time \( n \), the forecasts can be computed using residuals and the observations up to time \( n \). Point forecasts are usually computed most easily directly from the ARIMA model equation as explained further in this section, which Box et al [9] refer to as the difference equation form. The forecast is obtained from the model equation by replacing

i) present and past values of the random variable \( (X) \) and first difference \( (Z) \) by their observed values.

ii) future values of \( X \) by their conditional expectation. For a given time instant \( n \), the minimum square error forecast of \( x_{n+1} \) is expressed as

\[
\hat{x}_{n(t)} = E(x_{n+t}|x_n,x_{n-1},...) 
\]

For instance, considering a SARIMA\((1,0,0)^*(0,1,1)_{16}\) model, the model equation is:

\[
X_t = X_{t-16} + \alpha(X_{t-1} - X_{t-17}) + Z_t + \Theta Z_{t-16}
\]  \hspace{1cm} (4.4)

where

\[
\hat{x}_{t+1} = x_{t-15} + \alpha(x_t - x_{t-16}) + \Theta z_{t-15} \\
\hat{x}_{t+2} = x_{t-14} + \alpha(\hat{x}_{t+1} - x_{t-15}) + \Theta z_{t-14}
\]

Consequently the ARIMA algorithm for traffic prediction can be summarized by the following steps:
1) Model Abstraction which involves identification of trend and seasonality within the original series (Random Walk Model) [8], finally identifying the differencing order (d, D) for the trend and seasonal components. The first task is to difference the original series to induce stationary behaviour and identify the seasonal component from its autocorrelation structure. Since the autocorrelation structure is analysed without making any changes into the original data, it is referred to as Random Walk Model.

2) Model Design, which involves identifying the autoregressive (p,P), and moving average (q,Q) orders for general and seasonal patterns from the autocorrelation and partial autocorrelation patterns of original series.

3) Model Verification, which involves a validation check for residuals obtained after fitting the model with appropriate orders for AR and MA terms. If the model is valid, the residuals are random and have small variance.

4) Forecasting Equation, which involves generating the forecasts if the model is validated and the residuals are random.

Application of ARIMA for traffic prediction has been widely used in various fields of networks, such as forecasting and planning of NSFNET [10], the predictive congestion control for wide area networks [11] and predictive algorithms for dynamic bandwidth allocation [7]. My work will use seasonal ARIMA prediction of delay magnitudes within a networked control system.
4.2.2 Prediction and Diagnostic Checks

In order to check the efficacy of any model, it is necessary to perform a diagnostic test. The diagnostic checks can be grouped for:

1) Residual Analysis
2) Forecast Analysis

The statistics used for diagnostic tests are:

1) Standard Error: It is defined as the Standard Error of the estimate coefficients, the estimated variance of the coefficients.

2) PE/SE: Ratio of the point estimate of the actual coefficient with the standard error.

3) Mean Square Error (MSE): It is the expected square of the difference of actual and predicted values. Mathematically it can be specified as

\[ \text{MSE} = E(x_t - \hat{x}_t)^2 \]  

(4.5)

where \( E \) is the expected difference between the actual \( x_t \) and its estimate \( \hat{x}_t \).

The most common test used for residual analysis is the Portmanteau lack-of-fit test that uses the chi-square statistic. The test statistic is given as [7]

\[ Q = N \sum_{k=1}^{K} r_k^2 \]  

(4.6)

where \( r \) is the autocorrelation function for residuals, \( N \) is the number of terms in the differenced series, and \( K \) lies in the range 15 to 30. \( Q \) is the chi square statistic with \((K-p-q)\) degrees of freedom, \( p \) and \( q \) are AR and MA terms, respectively, for the model. If the tested model shows seasonal behaviour, it may be the case that the fitted model has not filtered the seasonality characteristics. For a seasonal model the residuals can be tested for Normalized Cumulative Periodogram. The Normalized Cumulative Periodogram is defined as [8]
Conceptualization of Temporal Specifications for Application to Distributed Real-Time Control Systems

\[ C(f_j) = \frac{\sum_{i=1}^{n} R^2(f_i)}{ns^2} \]  

(4.7)

where

\[ R^2(f_i) = \frac{2}{n} \left[ (\sum_{t=1}^{n} a_t \cos 2\pi f_i t)^2 + (\sum_{k=1}^{n} a_t \sin 2\pi f_i t)^2 \right] \]

\( f_i = i/n \) (the frequency) and \( 1/f_i \) = the period, \( a_t \) are errors, \( s^2 \) is error variance.

The \( R^2(f_i) \) specifies the adequacy of the goodness of fit of trigonometric functions with frequency \( f_i \) to the residuals. For instance, a seasonal residual span with lag 16, or frequency ratio (implying \( 1/f_i \) is 1/16) would show large values for \( R^2(1/16) \), \( R^2(1/32) \), \( R^2(1/48) \), etc. The adequacy of the model is tested through diagnostic checking. The diagnostic tool is a plot of the Cumulative Periodogram against the frequency \( f_i \). For a disturbance series, the plot of \( C(f_j) \) against \( f_j \) should be scattered about the straight line joining the points (0,0) and (0.5,1) [8].

The metrics like SE, MSE can be applied for both residuals and forecasts. Given the model and series data for time \( t_n \) (where \( n \) is the time index), the forecasts will involve the observations and residuals up to the time \( t_n \). The point forecasts can be generated directly using the model equation. Assuming that the model equation is definite, the forecast can be generated through the model equation by replacing [8]

a) future values of the differenced series by zero;

b) future values of the original series by their conditional expectation

c) present and past values of original and differenced series by their observed values.
The mean square error of the forecast $X_{t+h}$ at time $t$, $h$ steps ahead, will be the conditional expectation $X_{t+h}$ given as

$$E(X_{t+h}|X_t, X_{t-1}, \ldots) \text{ at time } t.$$  

(4.8)

The case study presented in the forthcoming section will analyse and use SARIMA$(1,0,1)*(1,1,0)^{16}$ and SARIMA$(0,0,2)*(1,2,1)^{62}$ models for the forecasts of Ethernet based DataSocket Transport Protocol (DSTP) delay magnitudes for the control data. For the adequacy of the fitted models, these diagnostic metrics are the benchmark for testing residuals and forecasts.

### 4.3 Experimental Setup for Communication Delay Modelling

#### 4.3.1 Protocols Analysis: DataSocket Transport Protocol (DSTP) & PROFIBUS DP

The networks that were modelled are Gigabit Ethernet with DSTP traffic and PROFIBUS DP. The traffic pattern of Ethernet is affected by the variable delays and requires more sophisticated modelling techniques for prediction than the traffic pattern of Control networks. DSTP is an application layer protocol used by DataSocket Server Manager [12]. Data Socket Server Manager is National Instruments’ propriety measurement data storage package [12]. DSTP is based on TCP/IP [12]. DSTP works on a Publish/Subscribe mechanism [9]. A system participating in a DSTP data exchange usually consists of three components – a publisher, the DataSocket Server and one or more subscribers [12]. A publisher transfers the process data from data acquisition devices to the server [12]. The server may be located on the same machine or remotely on a local network. Subscribers have to register interest to receive the updates from the server whenever the process data is revised [12]. The interesting factor that determines
the data transfer from publisher to subscriber is that the publisher broadcasts the entire active control dataset to a subscriber whenever any value changes, thus creating lot of network overhead. This can lead to poor scalability of the network as more and more subscribers become involved. Experiments were conducted to characterize the traffic and traffic prediction, using a SARIMA model for one to four subscribers.

PROFIBUS is a Fieldbus control network with real-time data transfer characteristics. It is one of the widely used control network for industrial automation, power generation, process automation and traffic engineering [13]. PROFIBUS complies with the IEC 61158 standard for measurement and control of digital data. The standard is titled “Fieldbus for Industrial Automation”. PROFIBUS uses layer 1, layer 2 and layer 7 of the ISO/OSI network model [13]. The PROFIBUS layered architecture is shown in Figure 4.1.

PROFIBUS uses the Bus Access Control procedure to sense the active stations for cyclic data exchange with corresponding stations [10]. It is a deterministic access method with real-time capabilities. The characteristic feature of PROFIBUS networks is the deterministic real-time communication cycle resulting from the use of master-slave principle [10]. Control data is communicated in a cyclic fashion, whereas diagnostic profiles, alarms and parameters are transferred in an acyclic manner. The PROFIBUS system can be controlled either by one or multiple master stations. With a single master station, that master communicates with each slave in turn and control data is transferred periodically over a network. Whereas in a multi-master system, the masters circulate the token for the control of the station. The master which possesses the token communicates
with the slaves and transfers the control data periodically. Acyclic communication occurs in parallel to the cyclic data transfer but with lower priority.

**PROFIBUS Layer Specification**

- **PA** – Process Automation
- **DP-V0, DP-V1, DP-V2**
- **FMS** – Fieldbus Message specification

---

**OSI Model**

- **Application Layer**
- **Presentation Layer**
- **Session Layer**
- **Transport Layer**
- **Network Layer**
- **Data Link Layer 2**
- **Physical Layer 1**

---

**PA – Process Automation, FMS – Fieldbus Message specification,**

**DP - Decentralized Periphery**

The PROFIBUS provides three communication protocols:

a) **PROFIBUS Fieldbus Message Specification (FMS):** This is specifically designed for communication at granular level between programmable logic controllers and digital processors. It supervises the transfer of structured data, control instructions and alarms etc.

b) **PROFIBUS Decentralized Periphery (DP):** This protocol is functionally classified into three versions namely DP-V0, DP-V1 and DP-V2. DP provides the basic functions, including the acyclic transmission of diagnostic profiles. DP-V1 is enhanced to enable the process automation with interrupt control of data acquisition devices. DP-V2 provides time-stamping and the
publish/subscribe mode of data transfer. It is applicable in low cost distributed control and automation systems, high-speed and time-critical applications.

c) PROFIBUS Process Automation (PA): This application layer protocol is designed for reliable and high speed communication. PROFIBUS PA extends the DP protocol for data transmission. It allows the simultaneous transmission of digital data through Manchester Encoding [13], bus powered (MBP) for hazardous areas. The protocol targets petrochemical and power plants.

The deterministic real-time communication guarantees the bus cycle time and bounded response time for the participating stations. The bus cycle time for the most basic PROFIBUS DP-V0 protocol is approximately 10 milliseconds. The transmission technologies for PROFIBUS networks are:

a) RS-485: This is an electrical transmission protocol based on shielded two wire cable with a transmission rate of 31.25 Kbits/sec. It can support 32 stations per segment.

b) RS-485-IS: This is intrinsically safe electrical transmission protocol for hazardous areas. The transmission rate is 1.5 Kbits/sec. The concept of Intrinsic Safety (IS) applies to safe operation of electronic equipments in explosive atmosphere or under disturbed operating environments. The RS485-IS intrinsically safe concept provides a PROFIBUS system with bounded maximum voltage and current. In set-up at least one PROFIBUS isolating repeater is located in the non-hazardous area for the safe separation of the intrinsically-safe bus segment from the non-intrinsically-safe bus segment [14]. Other connected communications devices (field devices) are
Conceptualization of Temporal Specifications for Application to Distributed Real-Time Control Systems

located in the hazardous area [14]. The bus cable is terminated at both ends by means of an external active bus termination or a bus termination integrated in a communication device as well as in a bus connector [14].

c) Fibre-Optic: This is optical network with linear, ring or star topologies for Optical Link Modules (OLM). It is preferable for large distance connections. It can support up to 126 stations.

PROFIBUS PA systems operating in hazardous environments make use of Manchester encoding, Bus Powered (MBP) with RS-485 IS communication. MBP is used in for intrinsic safety requirements of acquisition devices and hazardous environments. It can support 32 stations per segment.

4.3.2 Experimental Set-up For Ethernet and PROFIBUS DP

![Figure 4.2 Control Loop with Communication Links between Sensors and Actuators](image)

Figure 4.2 Control Loop with Communication Links between Sensors and Actuators

The experimental set-up for traffic modelling is shown in Figure 4.2. The control loop operation, closed through the communication channel, is used for modelling and predicting the delay magnitudes. The control algorithm for the second order non-linear water tank problem was considered and optimized to compensate for delays induced by
the network. The communication links selected were Gigabit Ethernet and PROFIBUS DP-V0. Ethernet traffic is non-deterministic and requires sophisticated modelling techniques whereas PROFIBUS traffic traces show bounded delay magnitude. Consequently ARIMA modelling [7] and forecasting was employed for Ethernet traffic. Experiments were performed, real traffic traces were obtained from DataSocket Transport Protocol (DSTP) operating over Gigabit Ethernet, for a maximum of four sets of subscribers. The capturing time varied from ten seconds to thirty seconds. Each sample traffic trace was collected over a time interval of 30 seconds for all sets of subscribers. The seasonality within the traffic pattern was judged by autocorrelation and partial autocorrelation charts of the data. The next section explains the detailed analysis, modelling and prediction of DSTP based Ethernet traffic.

The PROFIBUS traffic was captured from the set-up of peer-to-peer communication of two PLC’s. The data acquisition devices and PLC’s were linked using PROFIBUS DP-V0 links. The set-up was for base level communication where one PLC was programmed to simulate a plant transfer function and the second PLC was programmed to act as a controller. The transfer functions were implemented using STL version 2 for Siemens SIMATIC PLC S7-400. The bus cycle time was set to 10 milliseconds. The traffic was captured using a diagnostic tool called Advanced Multicard Profibus Analyzer (Amprolyzer). The diagnostic tool was a standalone application that logs message frames on the PROFIBUS. The PROFIBUS bus structure and experimental set up are shown in Figure 4.3.
4.3.3 Ethernet Traffic Characterization, Modelling & Prediction with ARIMA

Control loop operation, as explained in the previous section, was captured for one to four workstations and real traffic traces were captured over a period of 3 days continuously. The experiment was conducted in the Control Engineering Laboratory, School of Engineering and Energy, Murdoch University, with 5 workstations connected to the central server. The central Windows-based server uses DataSocket Server Manager as the control data repository. The real traffic traces were obtained, with students implementing the water-tank case study, as described in Chapter 3, on workstations for a period of 30 minutes each day, in 4 groups, with subscribers increasing from one to four. The arrival times (for requests from the subscribers and replies from the server) were captured and analysed for one to four workstations for DSTP traffic. The inter-arrival time gap samples were extracted from 4 datasets for a period of 30 seconds. Inter-arrival time is defined as the difference between the response packet arrival times for every request. There were 15 data response packets to
every subscriber’s request. The inter-arrival time increased with the number of workstations on the network. Therefore, to give detailed insight into the response time behaviour, in the presence of more than one subscriber, publication time was modelled. Publication time is defined as how long it takes for the subscribers get a response to their requests in the presence of other subscribers on the network. The first step is the characterization of the traffic using Publication Time Modelling. The variability of publication time is described as a Cumulative Distribution function (CDF) for 3 sets of subscribers. The real traffic traces of Ethernet traffic are shown in Figure 4.4b. The CDF’s for the 3 sets of samples are shown in Figure 4.5.

Figure 4.4a Control Engineering Laboratory at Murdoch University for Ethernet Traffic Modelling
As shown in Figure 4.4b, the variability in the inter-arrival gaps increases with the increase in the number of workstations. The inter-arrival gaps increase to 455 milliseconds in the 4 workstation configuration whereas, in one workstation dataset, it had a maximum value of 150 milliseconds. The descriptive statistics for the inter-arrival time gaps for three workstations are shown in Table 4.1.
Table 4.1. Descriptive Statistics for Inter-Arrival Time Gaps

<table>
<thead>
<tr>
<th>Workstations</th>
<th>Response Packet Count (over a period of 30 Seconds)</th>
<th>Mean (sec)</th>
<th>Variance (sec$^2$)</th>
<th>Standard Deviation (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>300</td>
<td>0.0065</td>
<td>0.0007</td>
<td>0.0026</td>
</tr>
<tr>
<td>3</td>
<td>848</td>
<td>0.0072</td>
<td>0.0342</td>
<td>0.0012</td>
</tr>
<tr>
<td>4</td>
<td>1192</td>
<td>0.0099</td>
<td>0.0434</td>
<td>0.001</td>
</tr>
</tbody>
</table>

Figure 4.5 Cumulative Distribution Function for Publication Time for One to Four Workstations

The graph in Figure 4.5 depicts the probability of delay variation from one to four workstations. The delay changes from a light-tailed distribution to a heavy-tailed distribution. As the number of clients increase on the network, the DSTP traffic congestion increases. The delay ranges from 10ms to 455 ms. The variation tends to increase as the number of workstations increase. The results are as predicted by the publish-subscribe mechanism; its major difficulty is that the technology often scales poorly, leading to heavily loaded networks. The delays manifest themselves as poor
throughput and slowdowns, as more and more applications use the system, thus saturating the network with overhead messages. This can make the system more prone to network delays. This is the obvious reason for increasing magnitude of delays as evident from the Figure 4.5. The variability in the delay distribution also increases with the increase in the subscribers. A summary for the average response time variation for four workstations is provided in Table 4.2.

**TABLE 4.2 Average Publication Time Delays (in milliseconds)**

<table>
<thead>
<tr>
<th>Workstations</th>
<th>Host1</th>
<th>Host2</th>
<th>Host3</th>
<th>Host4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>45</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>59</td>
<td>53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>67</td>
<td>67</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>98</td>
<td>92</td>
<td>104</td>
<td>98</td>
</tr>
</tbody>
</table>

Table 4.1 depicts the significant increase in the average publication time delay as the network becomes heavily loaded. For a single workstation the delay varies constantly over a period of 10 ms to 140 ms (this excludes the experiment control loop delay of 100 milliseconds). As the number of workstations increases, the publication frequency decreases and jitter is delay variation. The CDF tends to become a heavy-tailed distribution; therefore preliminary tests for the Hurst Parameter were estimated to check the degree of self similarity within the traffic.

The distribution of a random variable $X$ with distribution function $F$ is said to have a long tail if for all $t > 0$, $[15]$

$$\lim_{x \to \infty} \Pr[X > x + t | X > x] = 1,$$  \hspace{1cm} (4.9a)
or equivalently

\[ \bar{F}(x + t) \sim \bar{F}(x) \quad \text{as} \quad x \to \infty. \]  

(4.9b)

Generally, long-tailed distributions for the traffic characterize it as displaying long-range dependence and self-similarity. Self-similar traffic exhibits the same characteristics and behaviour when viewed at different time scales. Self-similarity in packet-based data networks can be caused by the distribution of file sizes, human interactions and Ethernet data transfer patterns [16]. “Self-similar and long-range dependent characteristics in computer networks present a fundamentally different set of problems to researchers doing analysis and/or design of networks, and many of the previous assumptions upon which systems have been built are no longer valid in the presence of self-similarity” [16]. But, assuming the probability distribution is heavy-tailed, it is not essential that the distribution of response gaps over time will show long-range dependence or self similarity, because all heavy-tailed distributions may not be long-tailed. The Hurst parameter, \( H \), is a measure of the degree of self-similarity of a time series. If the data is depicting self-similarity, \( H \) takes on values from 0.5 to 1. A value of 0.5 or less indicates the data is uncorrelated or that it has only short-range correlations and no self-similarity. The closer \( H \) is to 1, the greater the degree of persistence or long-range dependence [17]. The R/S statistic is a technique for estimating the Hurst parameter. It is discussed in [18] and [19]. Let \( R(n) \) be the range of the data aggregated (by simple summation) over blocks of length \( n \) and \( S(n) \) be the sample variance of the data aggregated at the same scale. The Hurst parameter is judged from the best line fit to the log-log plot of \( R(n)/S(n) \) versus \( n \).
As is evident from Figure 4.6, the Hurst parameter takes the value less than 0.5, indicating randomness within the data with short-term correlation structure. The heavy tailed CDF is the obvious result of traffic congestion as more workstations join the systems. The nature of the traffic is highly dependent on the application protocol in use.

Since DSTP uses a publish/subscribe mechanism, with more subscribers the network...
performance obviously degrades, indicating more deviation of delay magnitudes from their mean value.

The characterization of the traffic provides information about the traffic behaviour. Once the random behaviour of traffic is known, the performance analysis and hence delay predictions require sophisticated modelling techniques. Consequently, ARIMA modelling was performed for the two cases, i.e. one subscriber and four subscribers. Since the experiment had control loop delay of 100 to 150 ms, it was efficiently captured through seasonality in the autocorrelation structures of the random walk models of the respective samples. Therefore the delay prediction equation uses Seasonal ARIMA Modelling.

![Autocorrelation and partial autocorrelation of one workstation response time gaps](image)

a) Autocorrelation and partial autocorrelation of one workstation response time gaps
b) Autocorrelation and partial autocorrelation of four workstation response time gaps

**Figure 4.7 Random Walk Models for One and Four Workstations**

As is evident from Figure 4.7a, the random walk models depict the seasonal component that is identified at the multiples of time-lag 16, as there are fifteen response data packets transferred to the subscriber for every request. The response time behaviour induces a seasonal impact in the inter-arrival time gaps. The seasonal component is more obvious in the case of a single subscriber and dies out as more subscribers are added. Since the entire control data is broadcast to the subscribers, the transmission involves fifteen data packets of equal size. The fifteen data packets are transferred within the time range of 10 milliseconds to 450 milliseconds, thus causing seasonal impact on the autocorrelation patterns.

The autocorrelation patterns and partial autocorrelation patterns specify the order required by the SARIMA model. The best fits for both traffic patterns were obtained using Minitab 15 for orders as specified in Table 4.3. The residual autocorrelations and forecasts for both models are shown in Figure 4.8. Figure 4.8 shows that the residuals are random and close to zero. Consequently the fitted models were deemed adequate.
Forecasts obtained through ARIMA are based on specific assumptions that are supposed to be fixed for a particular set of experimental data.

Table 4.3. Fitted SARIMA Model Details for One and Four Workstation Response Time Gaps

<table>
<thead>
<tr>
<th>Workstations</th>
<th>Model Details for Response Time Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong></td>
<td><strong>SARIMA(1,0,1)*(1,1,0)_{16}</strong></td>
</tr>
<tr>
<td></td>
<td>Span(s): Time Lag 16</td>
</tr>
<tr>
<td></td>
<td>Final Estimates of Parameters</td>
</tr>
<tr>
<td>Type</td>
<td>Coef</td>
</tr>
<tr>
<td>AR(p)</td>
<td>1</td>
</tr>
<tr>
<td>SAR(P)</td>
<td>16</td>
</tr>
<tr>
<td>MA(q)</td>
<td>1</td>
</tr>
<tr>
<td>Constant</td>
<td></td>
</tr>
<tr>
<td>Differencing(d, D): 0 regular, 1 seasonal of order 16</td>
<td></td>
</tr>
<tr>
<td>Number of observations: Original series 443, after differencing 427</td>
<td></td>
</tr>
<tr>
<td><strong>Model Validation</strong></td>
<td></td>
</tr>
<tr>
<td>Chi-Square statistic</td>
<td></td>
</tr>
<tr>
<td>Lag</td>
<td>Chi-Square</td>
</tr>
<tr>
<td>12</td>
<td>0.2</td>
</tr>
<tr>
<td>24</td>
<td>11.1</td>
</tr>
<tr>
<td>36</td>
<td>28.9</td>
</tr>
<tr>
<td>48</td>
<td>52.6</td>
</tr>
<tr>
<td><strong>4</strong></td>
<td><strong>SARIMA(0,0,2)*(1,2,1)_{62}</strong></td>
</tr>
<tr>
<td></td>
<td>Span(s): Time Lag 62</td>
</tr>
<tr>
<td>Type</td>
<td>Coef</td>
</tr>
<tr>
<td>SAR(P)</td>
<td>62</td>
</tr>
<tr>
<td>MA(q)</td>
<td>1</td>
</tr>
<tr>
<td>MA(q)</td>
<td>2</td>
</tr>
<tr>
<td>SMA(Q)</td>
<td>62</td>
</tr>
<tr>
<td>Constant</td>
<td></td>
</tr>
<tr>
<td>Differencing(d, D): 0 regular, 2 seasonal of order 62</td>
<td></td>
</tr>
<tr>
<td>Number of observations: Original series 846, after differencing 722</td>
<td></td>
</tr>
<tr>
<td><strong>Model Validation</strong></td>
<td></td>
</tr>
<tr>
<td>Chi-Square statistic</td>
<td></td>
</tr>
<tr>
<td>Lag</td>
<td>Chi-Square</td>
</tr>
<tr>
<td>12</td>
<td>13.3</td>
</tr>
<tr>
<td>24</td>
<td>22.2</td>
</tr>
<tr>
<td>36</td>
<td>29.7</td>
</tr>
<tr>
<td>48</td>
<td>42.3</td>
</tr>
</tbody>
</table>
a) Residual and Fits for One Workstation Response Time Gaps

b) Residual and Fits for Four Workstations Response Time Gaps

**Figure 4.8 Residual and Forecast Analysis**

As seen in Figure 4.8, the residuals are close to zero and have small variance. Once validated, the model can also be used for predicting future inter-arrival times for the response packets. The forecast is generated using the model equation. The forecast model equations for the two models are as follows:
\[ X_t = X_{t-1} + 0.2384 (X_{t-1} - X_{t-16}) + Z_t + 0.5711 Z_{t-1} - 0.6390 X_{t-16} \]  
\[ \text{where } Z_t = X_t - X_{t-1} \]

(For 1 Workstation Model)

\[ X_t = -0.3682 X_{t-62} - 0.000056 (Z_t - Z_{t-62}) + Z_t - 0.0467 Z_{t-1} + 0.0161 Z_{t-2} + 0.7877 Z_{t-62} \]

\[ \text{where } Z_t = X_t - X_{t-1} \]

(For 4 Workstation Model)

The SARIMA models and the forecast analysis yield a stochastic delay model for network measurements. The essential aspect of network traffic is that it is application protocol specific. Consequently the delay modelling of application protocol using statistical models like Box Jenkins ARIMA can provide a better mathematical model of the response time behaviour over time. The modelling was successfully able to capture the periodic component within the traffic. Each subscriber which registers interest in the control data, the entire active dataset is published by the publisher. This yields 15 response packets that are transferred to the subscribers, causing seasonality within the traffic patterns. In a later chapter it will be shown how these models can be employed for designing predictable controllers that are optimized to take network delays into account.

### 4.3.4 PROFIBUS DP Traffic Analysis and Characterization

The control-loop operation was implemented using a PROFIBUS network transported by a RS 485 communication line between a PLC emulating plant behaviour and another PLC designed to be a controller. Real traffic traces were captured using Amprolyzer
over a period of 26.7 seconds. The bus cycle delay was set to 10 ms for one control loop operation.

![Figure 4.9 PROFIBUS DP Traffic Trace for period of 30 Seconds](image.png)

The round trip delays were less than ten milliseconds. The PROFIBUS traffic is deterministic and predictable. The characterization and modelling of control networks is more deterministic in comparison to Ethernet [20] and Device Net [21]. Medium access protocol (MAC) protocols generally fall in two categories [22]: random access and scheduled access. Networks that follow a random access MAC protocol use the carrier sense multiple access (CSMA) algorithm for collision detection and avoidance. Ethernet [20] uses CSMA/Collision detection (CD) and Device Net uses CSMA/Bitwise Arbitration (BA) algorithm. When a collision occurs, the traffic nodes will back off and wait for random amount of time before retransmission. Therefore the traffic traces are affected by variable delays as shown in the characterisation and modelling of Ethernet traffic.

On the other hand, control networks like PROFIBUS and FDDI [23] follow scheduled and deterministic access patterns. Generally, token passing (TP) and time division
multiple access (TDMA) algorithms [23] are used by the MAC layer. These protocols eliminate contention for the shared medium by allowing each node to follow a deterministic scheduled pattern of transmission. Therefore these networks are said to have scheduled access. Packet transmission delays in such networks can be bounded through token based periodic transmission.

4.4 Conclusion

The chapter focussed on modelling and characterisation of deterministic (PROFIBUS, FDDI, Token Ring, etc.) and non-deterministic (Ethernet, Device Net) networks. The research seeks to follow the a posteriori analysis of delay modelling and compensation for control-loop operation of a real-time control system. A posteriori analysis assumes that the network architecture is taken as it is and models the delay behaviour over time. With the advancement of network modelling and characterisation techniques, it is feasible to extract the nature of delays originating from the networks and represent them by statistical models. The modelling tools are stochastic approaches like ARIMA and probability distribution functions. The experimental analysis and results of modelling are shown for Ethernet and PROFIBUS. A stochastic model is used to compensate for the delay magnitudes explicitly for the control loop operations designed using predictive [24] and adaptive [25] algorithms. These models are the statistical representation of the delays originating from the communication medium. They can be incorporated into the control loop implementation to have a priori knowledge of the communication delays. Hence the predictive and adaptive controllers can be used to compensate their impact.
4.5 References

[14] Siemens, "Intrinsic Safety Control Drawing, Profibus Connector 6ES7972-0DA60-0XAO."
Chapter 5

Programmatic Rationale for Delay Compensation and Controller Design

Key Points

- Delay Compensation for Deterministic & Non-Deterministic Networks
- Model Reference Control
- Self-Tuned Controller
- Case Studies: Gigabit Ethernet, PROFIBUS DP

5.1 Introduction

Scheduling and timing are fundamentally important aspects for designing real-time control applications. This chapter will focus on design of pragmatic models for delay compensation within feedback control loops. In Chapter 4, statistical models for delays, which originate from deterministic and non-deterministic networks, were analysed. This chapter will use the statistical models that provide prior knowledge of the delays. Finally, the controller optimization will be achieved for the delays that originate from deterministic and non-deterministic networks. The computation delay is assumed to be bounded as the controller will be implemented using TDL. This chapter is divided into four sections. Section two provides a detailed tutorial on various control design methodologies such as Model Reference Control and Adaptive Control. Section three presents the programmatic model for delay compensation using Model Reference Control algorithms such as the Smith Predictor and Dahlin. These algorithms are
implemented in TDL to bound their computation delay for the control loop operation. Finally Section four explains the detailed design and implementation of an Adaptive Controller, Smith Predictor and Dahlin Controller for non-deterministic networks.

5.1.1 Mathematical Modelling for Networked Time-Delayed Control System

A well-designed discrete-time model of a system makes critical assumptions about the controller gain and sampling time. These critical assumptions are necessary because it is possible that, for the same controller parameters, a stable continuous-time system can become unstable when discretized. The critical assumptions for a closed loop system are embodied in the characteristic equation [1]:

$$1 + g_p(z)g_c(z) = 0$$

(5.1)

where $g_p(z)$ is the $z$-transform of a continuous plant transfer function and $g_c(z)$ is the discrete controller transfer function. For a discrete-time closed loop system to be stable, all the roots of equation (5.1), must lie within the unit circle [4]. One important factor that such a model does not take into consideration is the effect of random delays that arise from the communication medium embedded within a distributed control loop. The stability margins lay stress on retuning the controller parameters for a discrete-time system to take into consideration the impact of the analog-to-digital (A/D) and digital-to-analog (D/A) converter delays introduced within the control loop. In order to analyse the impact of delays that originate from a communication medium, the first step is to mathematically model the control system for a communication delay. For a discretized system, the delay is characterised by the relative magnitudes of the sampling periods
and the communication delay. The section below discusses two cases i.e. when delays are smaller than the sampling period and when delays are larger than the sampling period.

### 5.1.1.1 Delays Smaller than the Sampling Period

The first case to consider is when the communication delay is smaller than the sampling period ($\Delta t$). The control loop operation is delayed, but the delay is tolerable by the system.

![Figure 5.1 Control Signal Generation](image)

As is evident from Figure 5.1, the total loop delay is as the accumulation of the constituent delays around the control-loop:

$$\alpha_k = \alpha_{pc} + \alpha_{ca} + \alpha_{cd}$$ (5.2)
where $\alpha_{pc}$ is the network delay from plant to controller; $\alpha_{ca}$ is the network delay from the controller to an actuator and $\alpha_{cd}$ is the (bounded) computation delay compensated for using TDL. The delay $\alpha_{cd}$ is not shown in Figure 5.1, as it has negligible magnitude as compared to other two delays. A first-order system with $x$ as the state variable, $u$ as the manipulated variable, $y$ as the process variable and $\Delta t$ sampling period, is specified as [3],

$$x_{k+1} = \varphi x_k + \gamma_1(\alpha_k) u_{k\Delta t} + \gamma_2(\alpha_k) u_{(k-1)\Delta t}$$  \hspace{1cm} (5.3a)

$$y_k = C x_k$$  \hspace{1cm} (5.3b)

where

$$\varphi = e^{\tau \Delta t}, \quad \gamma_1(\alpha_k) = \int_0^{\Delta t - \alpha_k} e^{\tau s} ds, \quad \gamma_2(\alpha_k) = \int_{\Delta t - \alpha_k}^{\Delta t} e^{\tau s} ds, \quad C$$ is the matrix (linear) transform from state to process variable, $\tau$ – time constant for plant, $\Delta t$ – sampling period

The maximum tolerable delay can be estimated from the relative magnitudes of $\gamma_1$ and $\gamma_2$ by testing the roots of the resulting characteristic equation in Equation (5.1) [4].

Defining $S$ as the state vector for $x_k$, at discrete time index $k$, in Equation (5.4a), [3]

$$S = [x_k \hspace{0.5cm} u_{k-1}]^T$$  \hspace{1cm} (5.4a)

the closed-loop system becomes [3]

$$x_{k+1} = \tilde{\Phi}_k x_k;$$ \hspace{1cm} (5.5)

where

$$\tilde{\Phi}_k = \begin{bmatrix} \Phi - \gamma_1(\alpha_k) K & \gamma_2(\alpha_k) \\ -K & 0 \end{bmatrix}$$ \hspace{1cm} (5.6)
The system guarantees stability if the delays are bounded. Delays smaller than the sampling period can be well compensated for by the conventional one unit delay compensation techniques [1][11] because the state variable $x_k$ only depends on the value of $y_{k-1}$. The stability margins are established around $x_k$ and $\Delta t$ for a constant loop delay.

The method described in [3] provides an analytical approach to define the stability margins for sampling time and network delays that originate from deterministic networks. For a scalar system with sampling period $\Delta t$ and network delay of $\alpha$ [3, 5], the allowable bounds for $\alpha$ can be defined as

$$\max\{\frac{1}{2}\Delta t - \frac{1}{k}, 0\} < \alpha < \min\{\frac{1}{k}, \Delta t\}$$

Equation (5.7)

The system is capable of tolerating a delay of up to one sampling period. It may be infeasible to derive the exact stability region for general systems [5]. Equation (5.7) can be used to estimate the bounds for sampling time, given an estimate of the network delay. Considering equation (5.7), the stability bounds for the sampling time can be set experimentally for a particular control system, with a given values of network delay.

5.1.1.2 Delays Larger than the Sampling Period

The second case is when delay exceeds the sampling period, such that $\Delta t < \alpha < D\Delta t$; $D > 1$, there may be dead sampling period i.e. no signal is received. Defining $x_k$ as the state vector [3], (as explained in Section 5.1.1.1) for communication delays larger than the sampling period, it can be represented as shown in Equation (5.4b),

$$S = [x_k, u_{k-D}, \ldots, u_{k-1}]^T$$ (5.4b)
Since the presence of a communication delay merely causes a time shift in the response of the undelayed counterpart of the process, the delay step (D) is introduced, that signifies the magnitude of the shift. The time shift is always assumed to be an integer factor of the delay estimated within the control loop operation. If the communication delay is smaller than the sampling period, the time shift i.e. D is 1. If the communication delay is larger than the sampling period, the time shift is greater than 1.

5.1.2 Delay Step Estimation Algorithm

Smaller values of $\Delta t$ are often desirable if a discrete-time system is to achieve a close approximation to its continuous-time counterpart. However decreasing the sampling rate can increase the network load faced by a distributed control system, thus leading to less predictable delays in communication. Therefore, it is important to tune the sampling rate to achieve a balance between network load and desired system performance [5]. For instance, the critical assumption for the sampling rate for a first-order system, while converting it from continuous-time to discrete-time, can be given as [1]:

$$\Delta t_{\text{max}} = -\tau \ln\left(\frac{K K_c - 1}{K K_c + 1}\right)$$

(5.8a)

where $K$ is gain for the plant; $K_c$ is the controller gain for the PID; $\tau$ is the time constant for plant. For any values of $K$ and $K_c$, this implies

$$\frac{\tau}{10} \leq \Delta t_{\text{max}} \leq \frac{\tau}{4}$$

(5.8b)

If stable control loop operation is to be ensured, the sampling period should be smaller than $\Delta t_{\text{max}}$. The accurate judgement of maximum sampling rate here depends on the estimates of the magnitude of controller gain and time constant for the plant. For
choosing the sampling rate for such a time-delayed system, it is essential to violate the constraints given in Equation (5.7). Consequently, the sampling time is always assumed to be smaller than loop delay i.e. \( \alpha_k \), such that

\[
\alpha_k = D\Delta t; D = 1, 2, 3, \ldots
\]  

(5.8c)

The delay estimation algorithm is introduced, with the objective to evaluate the delay steps (D), given the magnitude of the communication delay. Finally the delay estimation algorithm can be summarized in the following steps:

**Listing 5.1**

For discrete time index \( k \) let \( \alpha_k \) be the round trip delay, measured within the control loop operation. The control data packets are time stamped. The difference between the time stamps for request and response packets, is defined as the round trip delay.

1. Estimate the Critical Limit Sampling Interval (\( \Delta t_{\text{max}} \)) such that

\[
\Delta t_{\text{max}} = -\tau \ln\left(\frac{K_k-1}{K_k+1}\right)
\]

2. Set the estimated sampling period (\( \Delta t \)) such that the condition \( \Delta t < \Delta t_{\text{max}} \) is met.

   // For Delay Steps to be greater than 1.

3. Repeat

   3. Set of past \( \alpha_k \) input vector as \([\alpha_k, \alpha_{k-1}, \alpha_{k-2}, \ldots]\) estimated from timestamp and buffered as round trip delays.

4. For simplicity, \( \bar{\alpha}_k \) is assumed to be the sum of controller to actuator delay and plant to controller delay, \( \bar{\alpha}_k = \alpha_{ca(k)} + \alpha_{pc(k)} \), since round trip delays are recorded from timestamps.

5. prediction for delay \( \bar{\alpha}_{k+1} = f(\bar{\alpha}_k) \) where \( f \) can be ARIMA generated model equation for Ethernet traffic modelling and estimated mean bounded value in case of deterministic networks.

   ARIMA Model Equation for implemented 1 Workstation Model:

   \[
   \bar{\alpha}_{k+1} = \alpha_{k-16} + 0.2384*(\alpha_{k-1} - \alpha_{k-2}) + Z_k + 0.5711*Z_{k-1} - 0.6390*\alpha_{t-16}
   \]

   where \( Z_k = \alpha_k - \alpha_{k-1} \) (Equation 4.10)

6. Set Delay Steps such that \( D_k = \frac{\bar{\alpha}_{k+1}}{\Delta t} \)

For each sampling period

The delay estimation algorithm finally generates the value for the delay step (D) that can be used for controller setting.
5.2 Model-Based and Adaptive Controller Design

5.2.1 Model Based Control

When the dynamic behaviour of a process model is predefined and a controller is derived to achieve a specified behaviour, the design paradigm is referred to as model-based control. The most commonly used model-based approach is Direct Synthesis Control, and the most basic form of model-based controller is the Smith Predictor [6]. When the nature of delays is known beforehand and they are bounded, the most basic controller that can minimize the impact of the time delay is the Smith Predictor. Time-delay compensation is achieved by introducing a minor feedback loop around the conventional controller. The minor feedback loop is referred to as a time delay compensator. The $G^*(s)$ is represented as the real process whereas $G(s)$ is the model process. The process signals for the real process and model process are shown in equations (5.9) and (5.10).

\[
y^*(s) = G^*(s)e^{-\alpha s} \tag{5.9}
\]
\[
y(s) = G(s)u(s) \tag{5.10}
\]

where $u(s)$ - control signal, $\alpha$ - time-delay within control loop

$y(s)$ - process output for the undelayed process.

The block representation of Smith predictor is as shown in Figure 5.2.

![Figure 5.2 Smith Predictor [1]](image-url)
The corrected error signal that reaches the controller after the introduction of minor loop is shown in equation [1] (5.11).

\[ e_c = y_c - y(s) - [y^*(s) - y(s)] \]  

(5.11)

where \( e_c \) is the corrected error signal.

Thus the minor loop eliminates the impact of time delay on the main feedback control loop where it can make the system unstable and is forwarded outside the loop. The closed-loop transfer function is shown in equation (5.12) [1].

\[ y = \left( \frac{G_c q_c}{1 + G_c q_c} \right) e^{-as} y_c \]  

(5.12)

The closed-loop transfer function as shown in equation (5.12), allows the use of conventional non-delayed controller without causing instability within the control loop. The Smith Predictor scheme works well for delays that are constant. However the design procedure for the Smith Predictor uses a conventional controller, yielding an unrealistic process output for the time-delayed process, because the minor feedback loop eliminates the delay impact and allows to have higher controller gains. The algorithm will work only if the process model is known and is without modelling errors i.e. is a precise representation of the physical plant.

The algorithm can be generalised by introducing the Direct Synthesis Controller. The Direct Synthesis Controller assumes that the closed-loop step response behaviour is predefined by a reference trajectory, \( q(s) \). The reference trajectory is chosen to guide the controller to follow the desired closed-loop behaviour. For a time-delayed process, the reference trajectory, as shown in Equation (5.12), allows the closed-loop behaviour to
be more realistic. The reference trajectory and first order time-delayed process transfer function can be specified as shown in Equation (5.13) [1].

\[
q(s) = \frac{e^{-\alpha r s}}{\tau_s s + 1} 
\]  
(5.13a)

\[
G(s) = \frac{Ke^{-\alpha s}}{\tau s + 1} 
\]  
(5.13b)

Given the process model and the reference trajectory, the controller, \( G_c \), is designed to achieve the desired closed-loop behaviour. The controller is redefined as shown in Equation (5.14) [1].

\[
G_c = \frac{(\tau s + 1)}{K} \left( \frac{e^{-(\alpha r-a)s}}{(\tau r+1) - e^{-\alpha r s}} \right) 
\]  
(5.14)

The control signal can be defined as in Equation (5.15) [1]:

\[
u(s) = \frac{r}{K\tau r} (1 + \frac{1}{\tau s})[e_c - [y^*(s) - y(s)]]
\]  
(5.15)

The design approach combines the Smith Predictor minor loop with a redefined controller using a reference trajectory, \( q(s) \). The discrete form for the control signal can be derived by substituting an approximation as shown in Equations (5.16a) and (5.16b)

\[
s \approx \frac{1 - z^{-1}}{\Delta t} 
\]  
(5.16a)

integer delay steps \( D = a/\Delta t \).  
(5.16b)

The time domain realization for the control signal is as shown in Equation (5.16c) [1].

\[
u(k) = \left[ \frac{\tau r + \Delta t}{\Delta t} \right] u(k - 1) + \left[ \frac{\Delta t}{\tau r + \Delta t} \right] u(k - D) + \left[ \frac{r + \Delta t}{k(\tau r + \Delta t)} \right] e(k) + \left[ \frac{r}{k(\tau r + \Delta t)} \right] e(k - 1)
\]  
(5.16c)

Direct Synthesis Control is only useful when the magnitude of the delay is constant. The controller has to emulate the desired closed-loop response and the process model needs to be perfectly known and defined. The zeros in the controller, as specified in Equation
(5.14), must perfectly cancel the poles in the plant, as specified in Equation (5.13). The
discrete-time controller is derived from the continuous-time domain and so the process
model is sensitive to the sampling interval. The choice of sampling interval is strictly
restricted to be greater than $\Delta t_{\text{max}}$ as shown in Equation (5.8). Therefore the design
approach requires a careful choice of the sampling interval for the discrete-time system
to closely emulate the continuous-time behaviour. Direct synthesis control is only
suitable for delay compensation when the loop delay is constant. Consequently, it is
more suitable for deterministic communication networks like PROFIBUS that have
bounded delay behaviour.

\subsection*{5.2.2 Dahlin Control Algorithm}

The class of controllers that are derived directly in the discrete-time domain are referred
to as direct controllers. There are several different digital controller design techniques
that differ only in their choice of reference trajectory in the discrete-time domain. Given
the process transfer function and desired reference trajectory as in Equation (5.13), and
assuming the delay to be an integral multiple of the sampling time (i.e. delay steps (D)
as $\alpha/\Delta t$), the discrete reference trajectory translates to the form shown in Equation
(5.17) [1].

$$q(z) = \frac{(1 - e^{-\Delta t/\tau z}) e^{-(D-1)}}{1 - e^{-\Delta t/\tau z} - 1}$$  \quad (5.17)

The Dahlin controller [7] can be specified as shown in Equation (5.18).

$$G_c(z) = \frac{\left(\frac{q(z)}{1 - q(z)}\right)}{G(z)}$$ \quad (5.18)

The controller depends on the inverse of plant’s transfer function, $G(z)$. The zeros of
$G(z)$ become the controller’s poles and vice versa. The introduction of a reference
trajectory in the closed-loop avoids the excessive control action due to the inverse of the plant transfer function. The two tuning parameters used in the Dahlin Controller are: the delay step (D) and time constant of the reference trajectory ($\tau_r$). The delay step (D) is chosen such that the process behaves realistically in the presence of the time delay and $\tau_r$ determines the speed of the closed-loop step response [1]. It makes the controller more realistic towards the time-delay. The controller is sensitive to the tuning parameters and is realizable if the delay is known a priori. Therefore it is the optimum choice for delay compensation within control networks, where the delay magnitude is bounded and known in advance.

The other form of controller design is adaptive control. For non-deterministic networks, where the delays are unpredictable, controller optimization to compensate for the delays can be achieved using adaptive control. The theory of adaptive control has developed using non-linear control theory with the aid of significant improvement in computer simulation in the last decade [8]. Adaptive control adapts itself to allow for system variations like changes in set-point, noise and disturbances. Hence, the strategy can be applied to compensate for unpredictable delays within non-deterministic networks, like Ethernet.

5.2.3 Adaptive Control

5.2.3.1 Introduction

Adaptive control systems [9] are designed using an adaptive controller structure that slowly changes its parameters over time according to the changes that occur within the plant model, to ensure the entire system behaves optimally in regard to the changing
aspects of the system. Generally, adaptive systems may be classified as: gain schedulers, model-reference adaptive systems and self-tuning systems [10].

Gain schedulers use an adjustment mechanism for the controller parameters according to process gain. It maps process parameters to controller parameters in the form of a function or table look-up [11]. Model reference adaptive systems specify their target performance in the form of a standard or a reference model [11]. The closed-loop process model aims to behave like a given reference model. The controller parameters are updated such that the error, which is the difference between the actual and model output, is minimised. Self-tuning regulators are indirect adaptive systems in which the controller parameters are adjusted and process parameters are estimated recursively [11]. Thus the controller and process parameters are updated at each sampling period. Since controller parameters are estimated and updated recursively, their estimation is done using either recursive least squares or ARMAX estimation methods (explained further in this chapter).

The self-tuned controller can be classified as explicit or implicit depending on the adaptation mechanism. If controller parameters are tuned directly, the controller is referred to as an implicit self-tuned regulator. In the explicit self-tuning, plant parameters are estimated and controller parameters are calculated and changed according to the estimated plant behaviour. Explicit self-tuning schemes are easy to design and are applicable to a wide spectrum of plants that are not required to be minimum phase [11] [12]. The recursive identification of the plant model may be achieved using recursive least squares, stochastic analysis or maximum likelihood
estimation techniques [11] [12]. The plant estimates, produced by these methods, are assumed to be accurate and are considered to have no estimation uncertainty. Consequently, the process is optimized using adaptive control techniques such as pole placement or minimum variance control [11]. The combination of estimation and control optimization leads to various self-tuned adaptive algorithms.

There is extensive published work on self-tuned controllers and a wide variety of estimation and adjustment mechanisms have been devised. Corripio and Tomkins designed the estimation algorithm using an instrumental variable method [13], whereas Wittenmark analysed pole placement techniques for application to digital PID controllers [14]. Kofahl and Isermann developed an adaptive algorithm using algebraic calculation of critical gain and critical periods of oscillation, followed by the use of the Zeigler-Nichols method [15]. BányáSZ et al. have suggested several modifications to the self-tuned structure for incorporation of the time delays into the feedback loop [11, 16]. In the late 1970s, work was done on self-tuned controllers based on pole assignment [17, 18].

5.2.3.2 Methodology

An explicit self-tuned adaptive system is dependant on the estimation mechanism for its plant behaviour. The general methodology is shown in Figure 5.3.
The plant and controller parameters are adjusted after each sampling period. The adaptive technique, introduced by Bányász and Keviczky [19], estimates time delay and adapts the process parameters automatically, using stochastic deterministic recursive adaptive schemes. If the delay estimation model is known a priori, the algorithm is implemented in three steps:

1. The delay is estimated using a stochastic model and both time lag and sampling interval are determined from Listing 5.1.
2. Process parameters are estimated using recursive least square estimation.
3. Controller parameters are tuned based on estimated plant behaviour.

Recursive plant parameter estimation is an essential step of the self-tuned adaptive system. Given the plant model in discrete-time, the parameters are updated recursively using the past and present plant input and output. If \( G(s) \) is the transfer function of a continuous time system, then the equivalent step response discrete-time transfer function with zero-order hold by applying \( z \)-transform is specified as [20]

\[
G(z) = (1 - z^{-1}) \mathcal{Z}\{G(s) \frac{1}{s}\}
\]  

(5.19)
for $\Delta t$ sample period.

The variable $q$ represents the forward time-shift operator [20], with $n$ as the degree of polynomial $A$, thus implying that

$$A^*(q) = q^n A(q) \quad (5.20)$$

Hence $A^*$ is the forward time shifted representation and $A$ is the back shifted form.

The predictor or estimator model can be designed using ARMAX or recursive least square estimator methods, as a function of previously measured variables and assuming it to be linear, specified as

$$y(k) = f(y(k-1), y(k-2), \ldots, y(k-n_a), u(k-1), u(k-1), \ldots, u(k-n_b), v(k-1), \ldots, v(k-n_d), e(k), e(k-1), \ldots, e(k-n_e), k)$$

or

$$A(q)y(k) = B(q) u(k) + D(q) v(k) + C(q) e(k) \quad (5.21)$$

where $v$ is the measurable component of the disturbance, and $e$ is the random non-measurable component disturbance [20].

The ARMAX model requires identification of the polynomial $C(q)$, as $e(k)$ denotes the random, non-measurable noise. This can make the estimator model clumsy and difficult to analyse. Therefore the preferred representation uses an ARX model, which redefines Equation (5.22) as

$$A(q)y(k) = B(q) u(k) + e(k) \quad (5.22)$$

With a Single Input Single Output system, the stochastic ARX model can be specified as [20]

$$y(k) = \Theta^t(k) \Phi(k-1)+ e(k) \quad (5.23)$$
where \( \Phi(k-1) \) is the data vector and \( \Theta^t(k) \) forms the parameters vector to be estimated.

\[
\Phi(k-1) = [-y(k-1), -y(k-1), \ldots, y(k-n_a), u(k-1), u(k-2), \ldots, u(k-n_b)]
\]

\[
\Theta^t(k) = [\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_{n_a}, \hat{b}_1, \hat{b}_2, \ldots, \hat{b}_{n_b}] \quad (5.24a)
\]

The unknown vector \( \Theta^t \) of dimension \( n \) can be estimated when minimizing the criterion [20]

\[
J_k(\Theta^t) = \sum_{i=0}^{k} e_i^2 (i) \quad \text{where} \quad e_i(i) = [i-\Theta^t] \left[ {y(i) \atop \Phi(i)} \right] \quad (5.24b)
\]

The unknown parameters in Equation (5.24a) and (5.24b) are calculated using recursive least square algorithm. Equation (5.24b) yields the solution to the estimated plant parameters \( \hat{A} \) and \( \hat{B} \). The plant model with estimated parameters can be specified as shown in Equation (5.25).

\[
\hat{A}(q)y(k) = \hat{B}(q) u(k) + e(k) \quad (5.25)
\]

where

\[
\hat{A}(q) = q^{n_a} + \hat{a}_1 q^{n_a-1} + \ldots + \hat{a}_{n_a}
\]

\[
\hat{B}(q) = \hat{b}_1 q^{n_b} + \hat{b}_2 q^{n_b-1} + \ldots + \hat{b}_{n_b}
\]

The controller is designed and derived from the estimated plant model. The algorithmic structure varies depending on the tuning procedure. The stochastic behaviour of the delay should be known a priori. The delay estimation algorithm can be used to simulate the delay magnitude within the control loop. The controller implementation is done using a discrete filter that is serially connected with the plant within a control loop. The
number of delay steps is estimated from the delay estimation algorithm, and delay steps (D) varies with the magnitude of the delay simulated within the control loop. The discrete transfer function for the controller and delay steps according to the magnitude digital filter are specified [20]

\[ G_c(z) = \frac{p_0 + p_2 z^{-1} + p_2 z^{-2}}{1 - z^{-1}} \]  \hspace{1cm} (5.26a)

\[ G_{df}(z) = \frac{1}{1 + \gamma z^{-1}} \]  \hspace{1cm} (5.26b)

The time delayed plant transfer function is specified as shown in equation (5.27).

\[ G_p(z) = \frac{b_0 (1 + \gamma z^{-1})}{1 + a_1 z^{-1} + a_2 z^{-2}} z^{-D} \]  \hspace{1cm} (5.27)

where D = α/Δt;

The controller assumes prior knowledge of the delay step. The delay estimation algorithm estimates the delay entering the control loop. The parameter estimate required for the self-tuning controller and the regression vector takes the form

\[ \Theta^c(k) = [\tilde{a}_1, \tilde{a}_2, \tilde{b}_0, \tilde{b}_1] \text{ and } \phi^c(k-1) = [-y_{k-1}, -y_{k-2}, u_{k-D}, u_{k-D-1}] \]  \hspace{1cm} (5.28)

The Bányász and Keviczky Adaptive algorithm compensates for the delay magnitude using parameter estimation and hence adaptively optimizes the controller tuning parameters. Having obtained the estimated process parameters, the PID regulator parameters can be computed by the application of the following formulae [11]

\[ \gamma = b_1/b_0 ; \]  \hspace{1cm} (5.29a)

\[ K_i = \frac{1}{2d-1} \text{ if } \gamma < 0; \]  \hspace{1cm} (5.29b)

\[ K_i = \frac{1}{2d(1+\gamma)(1-\gamma)} \text{ if } \gamma > 0 \]  \hspace{1cm} (5.29c)
From Equations (5.26-5.29), the controller can be finally realized by connecting a
digital filter serially in the control loop as shown in Figure 5.4.

Figure 5.4 Control Loop for Self-tuned Bányász and Keviczky Adaptive Algorithm

Adaptive controllers perform better when process parameters are subjected to periodic
re-estimation. Process identification is an integral component in designing adaptive
trollers. The tuning procedures thereafter adapt the system to changing process
characteristics. An adaptive controller may compensate for delays that originate from
non-deterministic networks with their magnitude determined using an appropriate
statistical model. The work presented here assumes a priori knowledge of the traffic
patterns of the networks under consideration.

5.2.4 Metrics for Closed-Loop Response Analysis

The Smith Predictor, Dahlin Controller and Self-tuned Adaptive algorithms were
analysed to compensate for control loop delays when communicating over
deterministic and non-deterministic networks. The metrics for closed-loop response
analysis specify how quickly the system responds to set-point changes. The two metrics
considered were the minimum settling time and the maximum percentage overshoot
brought about by a step input. Step input refers to the change in the output behaviour of
system when the input instantaneously changes.
Overshoot refers to an output that exceeds its final steady-state value. Percentage overshoot (PO) is calculated as the ratio of the difference between maximum and final value to the final value.

Settling time is defined as the time required by the system to settle within ten percent $\pm 10\%$ of the of the steady-state value.

The diagrammatic representation of metrics discussed above is shown in Figure 5.5.

Figure 5.5 Closed-Loop Response Metric Analysis

5.3 Delay Compensation Algorithms for Deterministic Networks

Communication delays that are bounded do not require complex controller tuning procedures for their compensation. The case study for PROFIBUS DP, discussed in Chapter 4, is used as an example for delay compensation of deterministic networks. The control-loop operation was designed and executed in real-time using TDL and simulated in the TDLVisualCreator/SIMULINK environment. The computation time is bounded as the control algorithm is implemented in TDL. The communication delay is assumed
to be bounded with the maximum value of ten milliseconds (this magnitude is derived from the experimental set-up for PROFIBUS DP). Both the Smith Predictor and Dahlin Controller were used to control the plant. A simple example of a second order system was considered, that maintains the water flow in and out of two non-interacting tanks, with a strict timing constraint to reach a given height of the water level. The aim was to analyse the control problem for real-time requirements and optimize the system to achieve these requirements. The two water tank case study as second order process with continuous transfer function given as

\[ G_{pd}(s) = \frac{1}{(2.618s+1)(0.382s+1)} e^{-0.018s} \] (5.30)

Listing 5.2 is the Smith Predictor in TDL/Java. The real-time code is in two modules. The main module is the timing specification and the second module is a Java class designed as a wrapper to the functional code.

**Listing 5.2**

```java
module Controller /* Timing Specification */
{
    task ControlOperation[wcet=30ms]
    {
        uses SmithPredictor();
    }
    start mode main[period=100ms]
    {
        task[freq=1] ControlOperation();
    }
}

class SmithPredictor /* Functionality Specification */
{
    public static void SmithPredictor()
    {
        /* Declaration of the Data */
        double sp, K, KC, deltat, tao, taoi;
        sp=0.37;
        double h[] = new double[50000];
        double u[] = new double[50000];
        double e[] = new double[50000];
        double deltau[] = new double[50000];
        double time[] = new double[50000];
        /* Initialization of Data deltat = Δt; D = delay steps Estimated from Listing 5.1 i.e. Delay estimation algorithm */
```
The control loop operation was simulated in TDLVisualCreator/Simulink. The worst case execution time was 30 ms and communication delay is 10 ms. The choice of period is made arbitrarily. Assuming the plant and sensor delay to be one second, the total delay induced in the control loop operation was 1040 ms. Figures 5.6a and 5.6b show the execution and communication delay compensation with TDLVisual Creator/Simulink. As evident in Figure 5.6a, the worst case computation time for a single control loop iteration was 30 milliseconds. TDL bounds the computation time. Figure 5.6b explains the process output before and after delay compensation with a Smith Predictor. The process output shows jitter and instability in the closed-loop response without delay compensation. The uncompensated experiment is truncated after two seconds because this system does not reach a stable final value.
Figure 5.6a Computation of Controller Module in 30 ms

Listing 5.3 shows how the Dahlin control algorithm was designed in TDL/Java. The real-time code was designed as two modules. The main module was the timing specification and the second module was Java class designed as a wrapper to the functional code.

Listing 5.3

```java
module Controller /* Timing Specification */
{
    task ControlOperation[wcet=30ms]
    {
        uses DahlinControl();
    }
    start mode main[period=100ms]
    {
        task[freq=1] ControlOperation();
    }
}
```
The control loop operation was simulated in TDLVisualCreator/Simulink. The computation time was 30 ms and communication delay was 10 ms. Assuming the data acquisition delay to be one second, the total delay induced in the control loop operation
was 1040 ms. Figures 5.7 and 5.8 show the computation and communication delay compensation with TDL Visual Creator/Simulink.

![Dispatch Table for module Controller](image)

**Figure 5.7 Computation of Controller Module in 30 ms**

![Figure 5.8 Dahlin Control](image)

a) Control Loop Without Delay Compensation  

b) Dahlin Algorithm with Delay Compensation

Communication and computation delay compensation is not difficult in the case of deterministic or control networks because the delay magnitude is bounded. For PROFIBUS DP, the average delay bound is taken as the communication delay metric. The Smith Predictor and Dahlin control algorithms were designed in TDL to implement the control loop behaviour. The Dahlin control algorithm was implemented directly in discrete-time and uses a less aggressive reference trajectory than the Smith Predictor, to have a more realistic view of the process behaviour in the presence of the time delay.
Table 5.1 summarizes the computation time and control loop settling time for both algorithms as noted from Figures 5.6 and 5.8.

Table 5.1 Response time estimate and computation time in TDL/SIMULINK

<table>
<thead>
<tr>
<th>Reference Trajectory (3 Minutes)</th>
<th>Non-Delayed Continuous-Time System</th>
<th>Smith Predictor (Discrete-Time)</th>
<th>Dahlin Algorithm (Discrete-Time)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TDL Computation Time(ms)</td>
<td>TDL Computation Time(ms)</td>
<td>TDL Computation Time(ms)</td>
</tr>
<tr>
<td></td>
<td>Settling Time(sec)</td>
<td>Settling Time(sec)</td>
<td>Settling Time(sec)</td>
</tr>
<tr>
<td>Control Loop in Simulink</td>
<td>20</td>
<td>2.4</td>
<td>30</td>
</tr>
<tr>
<td>Control Loop in TDL</td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30</td>
</tr>
</tbody>
</table>

As shown in Table 5.1, the discrete implementations are able to compensate for the delay, thus guaranteeing the stability of the control loop in the presence of unpredictable delays. However, in practice, the settling time may be high. The computation time for non-delayed system, Smith Predictor and Dahlin control algorithm was analysed by implementing the control algorithms and testing their execution in TDLVisualCreator. Settling time denotes the time taken by the process signal to become stable. Secondly, in both strategies, the delays are treated as bounded. These techniques are applicable in the case when delays are constant or bounded. But in the case of non-deterministic networks such as Ethernet, the delays cannot be bounded and are random. These techniques need to be applied adaptively to compensate for the changing magnitude of the delay. Further, a precise statistical model is required that can perfectly emulate the
behaviour of traffic within such networks. In Chapter 4, ARIMA modelling was used to model the Ethernet/DSTP traffic. The modelling results will be used in the next section to compensate for the delays within a control loop linked through Ethernet.

5.4 Delay Compensation Algorithms for Non-Deterministic Networks

Non-deterministic networks such as Ethernet exhibit random traffic patterns and the delay magnitude is not bounded. Consequently, such networks require precise stochastic models to emulate their traffic patterns. As explained in Chapter 4, Seasonal ARIMA modelling was performed for Ethernet/DSTP traffic. This model will be used as an example for the delays originating from non-deterministic networks. The computation time is bounded as the control algorithm was implemented in TDL. The delays were derived from a SARIMA(1,0,1)*(1,1,0)_{16} model of one workstation and a SARIMA(0,0,2)*(1,2,1)_{62} model of four workstations. The delay estimation algorithm was implemented with one workstation and four workstation model equations to generate the delay estimate for next sampling period. The model equations are as specified in Equations (5.31a) and (5.31b).

\[
\alpha_t = \alpha_{t-16} + 0.2384*(\alpha_{t-1} - \alpha_{t-17}) + Z_t + 0.5711*Z_{t-1} - 0.6390*\alpha_{t-16} \tag{5.31a}
\]

where \(Z_t = \alpha_t - \alpha_{t-1}\) (For 1 Workstation Model)

\[
\alpha_t = -0.3682*\alpha_{t-62} - 0.000056*(Z_t - Z_{t-62}) + Z_t - 0.0467*Z_{t-1} + 0.0161*Z_{t-2} + 0.7877*Z_{t-62} \tag{5.31b}
\]

where \(Z_t = \alpha_t - \alpha_{t-1}\) (For 4 Workstation Model)

The next step is to optimize the control loop operation with a known value of delay step (D) to be introduced. The control loop was optimized using three algorithms and performance was compared. The Ethernet traffic pattern is random, and for every
control loop iteration, the predicted value for the delay magnitude will change and hence the value of the delay steps will vary. Therefore Smith Predictor and Dahlin Control was implemented as a cascaded controller with three controllers optimized for different bounds on D.

As explained in Chapter 4, the DSTP traffic pattern was emulated by the SARIMA models, the delay is categorized in three groups: the average recurring delay, the best recurring delay and the worst recurring delay. This allowed prediction of the current estimation of network delay $a_k$ on based past measurements. From the traffic categorization and SARIMA generated forecasts, 60% packets had response time less than ten milliseconds, 25% had the response time ranging from 20-100 milliseconds and 15% had response time ranging from 300-500 milliseconds. TDL/Simulink models were generated for the Smith Predictor, Dahlin algorithm and Adaptive Controller with varying delay steps. Again the example of a second order system was considered. The aim was to analyse the control problem for real-time requirements and optimize the system to achieve these requirements. Considering the second order process with discrete-time transfer function and continuous-time given as

$$G_{pd}(z^{-1}) = \frac{0.0011z^{-2}}{1 - 1.966z^{-1} + 0.966z^{-2}} z^{-D} \quad (5.32a)$$

$$G_{pd}(s) = \frac{1}{(2.618s+1)(0.382s+1)} e^{-\alpha s} \quad (5.32b)$$

Time delay within control loop ranging over $1040 \text{ ms} < \alpha < 2400 \text{ ms}$;

Delay Steps ranging from $2 \leq D \leq 7$ as from Listing 5.1.

Figure 5.9 shows the cascaded control algorithm developed using a Smith Predictor.

The PID Controller was tuned for average, best and worst case delays. The module was
developed in three steps. The model in Figure 5.9(a) is the Simulink control loop for a plant, with controller and delay magnitude encapsulated in the subsystem. The subsystem is expanded in Figure 5.9(b). The controllers were tuned separately for the average, best and worst case delay magnitudes. The TDL module emulates the minor feedback loop for the Smith Predictor. Hence the Smith Predictor minor loop is implemented as three different TDL modules, tuned for different delay magnitudes. In order to ensure the integrity of the control signal within three TDL modules the rate transition block was added to Figure 5.9(a). The TDL module is expanded in Figure 5.9(c). The control algorithm was executed with 30 milliseconds as the bounded computational delay. The process and the control output is shown in Figure 5.10.

![Simulink Model for Control Loop Operation with Ethernet/DSTP Communication Link](image)

a) Simulink Model for Control Loop Operation with Ethernet/DSTP Communication Link
b) Smith Predictor Subsystem Expanded

c) One TDL Module Subsystem using TDLVisualCreator

Figure 5.9 Smith Predictor for Varying Delay Magnitude within Control Loop
As evident from Figure 5.10, the system reaches the final set-point within 11 minutes.

The Smith Predictor implementation is suitable only when delays are bounded. Moreover this algorithm only applies to the worst-case delay value, which has a rare probability of occurrence. Thus it can act as a major loophole for stability in the discrete-time domain. A realistic system with time delay is inherently unable to respond instantaneously to a control event. The high controller gains fail to provide a realistic image of the process. The other drawback is the lack of adaptability to changing process behaviour. The delay magnitudes are changing randomly, and are being emulated using a stochastic model. The delay changing pattern, as generated by SARIMA model is not taken into account, because the delays are finally grouped as average, best and worst case magnitudes. This assumption violates the actual delay generation patterns within the control loop and assumes the delay magnitudes to be bounded.
The next implementation used the Dahlin Control Algorithm. The Dahlin Control algorithm was implemented as a TDL/Simulink Module. The computation delay was assumed to be 30 milliseconds for single control loop iteration.

![Dahlin Control Algorithm Diagram](image)

**a) Dahlin Control for Delay Compensation**

![TDL Module Expanded Diagram](image)

**b) TDL Module Expanded**

**Figure 5.11 Dahlin Algorithm for Varying Delay Magnitude within Control-Loop**

The Dahlin control algorithm was tuned for a realistic time-delayed reference trajectory. As before, in order to ensure the integrity of the control signal within two TDL modules the rate transition block was added to Figure 5.11(a). The process output and control signal generated after a step input from the algorithm execution are shown in Figure 5.12.
Conceptualization of Temporal Specifications for Application to Distributed Real-Time Control Systems

Dahlin control performed better than the Smith Predictor. But the drawback is the lack of adaptability to the changing delays. The delay magnitudes are changing randomly and are being emulated using a stochastic model. The delay behaviour is characterised using a SARIMA model and in order to incorporate the changing delay pattern originating from communication link, an adaptive self-tuning procedure was chosen for delay compensation.

The adaptive self-tuned algorithm adapts to the changing process behaviour by recursively estimating the process parameters and hence re-tuning the control parameters for each control loop iteration. As in this case study, the adaptive self-tuned controller is executed in three steps as shown in Listing 5.4.

**Listing 5.4**

1) Estimate $\Delta t$, $D$ from delay estimation algorithm as explained in Listing 5.1. Listing 5.1 is modified as the process identification is performed for every sampling period and plant parameters change.

   a. Set of past $\alpha_k$ input vector as $[\alpha_{(k-1)}, \alpha_{(k-2)}, \alpha_{(k-3)}, ....]$ estimated from timestamp.
b. For simplicity, \( \alpha(k) = \alpha_c(k-1) + \alpha_p(k) \) since round trip delay is measured using timestamps.

c. \( \alpha(k+1) = f(\alpha(k)) \) where \( f \) can be ARIMA generated model equation for Ethernet traffic modelling

ARIMA Model Equation For 1 Workstation Model:

\[
\alpha_{k+1} = \alpha_{k-16} + 0.2384*(\alpha_{k-1} - \alpha_{k-17}) + Z_k + 0.5711*Z_{k-1} - 0.6390*\alpha_{k-16}
\]

where \( Z_k = \alpha_k - \alpha_{k-1} \)

d. Estimate the Critical Limit Sampling Interval (\( \Delta t_{\text{max}} \)) such that

\[
\Delta t_{\text{max}} = -\tau \ln\left( \frac{K_k-1}{K_k+1} \right)
\]

e. Set \( \Delta \tilde{t} \) such that \( \Delta \tilde{t} < \Delta t_{\text{max}} \) // For Delay Steps to be greater than 1.

f. Delay Steps (D) = \( \alpha_{k+1}/ \Delta \tilde{t} \)

2) A) Assuming prior knowledge of the number of delay steps form Step1, the Parameter Estimate vector for recursive ARX plant estimation is

\( \Theta^i(k) = [\alpha_1, \alpha_2, b_0] \)

a) The observation vector is assumed to be

\( \Phi(k-1) = [-y(k-1), -y(k-2), u(k-D)] \)

3) Tune the controller with discrete transfer function

\[
G_c(z) = \frac{p_0 + p_1z^{-1} + p_2z^{-2}}{1-z^{-1}}
\]

using Bányász and Keviczky tuning rules

\[
K_1 = 1/(2*D-1)\]

\[
p_0 = K_1/b_0;\]

\[
p_1 = (K_1^*\alpha_1)/b_0;\]

\[
p_2 = (K_1^*\alpha_2)/b_0;\]

4) Repeat Steps(1-3) for each sampling period.

The algorithm was implemented as three TDL Modules. One module implements the stochastic SARIMA Delay model. The second module is the recursive ARX plant parameter estimation. The third module is the Controller tuning based on plant estimation. The TDL specification for the modules is shown in Listing 5.5.

**Listing 5.5**

```plaintext
module Controller /* Timing Specification */ {
    task PlantEstimator[wct=30ms]
    { 
        uses PlantEstimation();
    }
    task SelfTuning[wct=20ms]
    { 
        uses ControlTuning();
    }
    start mode main[period=100ms]
```
The TDL/Simulink Models for the Adaptive Self-tuned controller are shown in Figure 5.13. The Plant model is estimated and hence the controller parameters are retuned for each sampling period. The communication delay is simulated into the control loop using Stochastic modelling SARIMA model for one workstation.

**a) Adaptive Self-Tuned Control Block in TDL/SIMULINK**

**b) ARX Plant Estimation TDL Module expanded**

Figure 5.13 Adaptive Self-Tuned Regulator
As is evident from Figure 5.14, the adaptive controller performs better in comparison to the discrete-time algorithms like the Smith Predictor. The random delay magnitude is fed into the loop as a Matlab M-Function using SARIMA model equations. The delays are fed into the control loop and buffered using time-stamps for the next predicted delay. The plant parameters are calculated for a sufficiently long period and the
covariance matrix is initialised for the first 30 iterations to provide better estimates prior to activating the controller. The control signal shows severe oscillation for 30 sampling periods because the controller is introduced after 30 iterations. The plant parameter estimates are shown in Figure 5.14(b). The behaviour of plant estimates and controller output depend on the initialization of the covariance matrix. As can be seen in Figure 5.14(b), the plant estimates converge. The oscillation is noticed when the controller is introduced into the control-loop after having sufficient initialization to estimate the covariance matrix. It is obvious, that the controller adapts the process to changing delay magnitudes. The changed set-point was introduced at the 80th, 100th and 150th sampling periods. The Process signal adapts to the changing set-points successfully.

The case study implements an indirect self-tuning regulator. The indirect algorithms are a straightforward implementation in that process parameters are estimated and controller parameters are computed by using Bányász and Keviczky’s [17] design equations [20].

### 5.4.1 Comparative Analysis for Smith Predictor, Dahlin and Self-Tuned Regulator for Random Delay Compensation

In section 5.4, three real-time implementations namely the Smith Predictor, Dahlin Controller and Self-Tuned Regulator were implemented for compensating random delays originating in non-deterministic networks. This section compares process signal and real-time implementation for the three algorithms. The results are summarized in Table 5.2.
Table 5.2 Response Time Estimates

<table>
<thead>
<tr>
<th></th>
<th>Smith Predictor</th>
<th>Dahlin Control</th>
<th>Self-Tuned Regulator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TDL Computation Time (ms)</td>
<td>% Overshoot</td>
<td>Settling Time (Min)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>86</td>
<td>11</td>
<td>30</td>
</tr>
</tbody>
</table>

As seen in Table 5.2, the real-time implementation for the self-tuned regulator requires 50 ms due to the recursive plant parameter estimation for each sampling period. But compared to the Smith Predictor and Dahlin Algorithm, the self-tuned controlled process settles much faster and is more adaptable towards the changing delay magnitudes.

5.5 Conclusion

Scheduling and timing are fundamentally important aspects of real-time control applications. It is a major challenge to provide an effective implementation that fits all aspects of real-time control systems. The complex structure of industrial plants, large production volumes and diverse interconnected units in operation led to the development of Model Predictive Control [6]. The chapter presented the real-time implementations of the control algorithms with delay compensation for two different communication links. The communications links analysed were PROFIBUS DP from the domain of Industrial Automation Networks, and Ethernet from the domain of Non-Deterministic networks. The delay compensation and real-time code were successfully implemented using Timing Definition Language (TDL). The objective of designing the control algorithms in a RT-Language is to bound the computation delay that originates...
from digital processing. Finally the real-time implementation for a self-tuned regulator was explained. The key factor for using adaptive control is the efficiency and adaptation for variations within the process dynamics. The variation within the process dynamics is due to the random delays that originate from the non-deterministic communication links. Having the appropriate stochastic model for the delay behaviour for a non-deterministic network, the delays can be successfully compensated using adaptive control design. The application of this work is strongly recommended for hard real-time control applications, where the delay magnitude of the order of a millisecond can hamper the stability of the system.

5.6 References

Key Points

Validation for Research Objectives

Future Work

6.1 Conclusion

This thesis has met the following objectives:

a) It has demonstrated the programmatic techniques for minimizing the impact of delays within the control loop.

b) Secondly, it has modelled and analysed the impact of network-induced delays by industrial automation network protocols. The case studies examined were deterministic networks (PROFIBUS DP) and non-deterministic networks (Gigabit Ethernet). The application protocol analysed and modelled was DataSocket Transport Protocol (DSTP).

c) It has analysed and compared various predictive and adaptive control algorithms for their stability capabilities with respect to time-delays.

d) It has explored and reviewed the benefits for application of RT-Languages to designing control applications.

e) Finally, it has compensated the communication and computation delays to provide a stabilized control behavior through predictive and adaptive control techniques, for enhanced real-time characteristics of a control loop operation within a plant model.
The project explored the application of modern tools and software technologies to industrial control systems. Specifically, the application of real-time characteristics to design issues of industrial control systems. Much advancement was achieved towards software tools and meeting real-time constraints within embedded systems. The real-time constraints become more obvious in hard real-time control systems. Hard real-time control systems are industrial control systems, that have response time specified as an absolute value that cannot be violated. Even very low probability of time violation can lead to destabilization of the system. Examples of such systems are aircraft flight control, anti-lock break systems within cars and jet engine control systems. My thesis applies in this domain. The time-delayed systems were the subject of research and RT-Languages were analysed to validate their applicability to this domain.

The evolution and transitional developments of Real-Time Languages were analysed deeply and finally Time Definition Language (TDL) was chosen as the language for designing and optimizing control algorithms. The digitized control-loop operation gives a new notion for time delays within a system. This is analysed in the first chapter. The main objective of the thesis is to optimize control performance in the presence of computational and communication delay by the application of a modern set of Real-Time tools and technologies.

Firstly, the requirement is to model the sources of the delays and have appropriate mathematical and stochastic models, which can be used for delay compensation and hence optimizing control performance. The case studies chosen for communication
delay modelling are Profibus DP from the domain of control networks and ethernet amongst the commonly used networks. The delay behaviour was tested experimentally for National Instrument’s DataSocket Transport Protocol (DSTP). The protocol structure is reviewed in Appendix ‘B’. The experimental data for inter-arrival times of response packets was recorded and analysed. The sample table of inter-arrival gaps between response packets for 1 workstation is given in Appendix ‘C’. ARIMA modelling is performed for Ethernet traffic. The details of the DSTP analysis and the API designed for data transfer are given in Appendix ‘B’.

After identifying the stochastic model for the delay patterns that originate from Profibus DP and Ethernet, the next task is to determine the appropriate predictive control algorithms for compensating the delays within control loops. The algorithms selected were, the Smith Predictor and Dahlin Controller for Profibus DP communication link and Adaptive Control for Ethernet communication link. The reasons for the above choice are:

a) Control networks such as PROFIBUS were deterministic in the delay generation patterns. The delays in such networks were bounded.

b) Secondly, compensating a bounded delay is straightforward and control loop optimization can be easily handled using direct predictive control algorithms such as the Smith Predictor and discrete-time algorithms such as Dahlin Controller.

c) Thirdly, the delay pattern for Ethernet is random. The delay simulated within the control loop operation hence changes and the process dynamics also change. Consequently, direct predictive algorithms are not able to adapt to the changing
magnitudes of the delays entering the control loop. There is a requirement for an
delay that can accommodate for the changing delay and
hence the process behaviour. The adaptive self-tuned regulator is finally chosen
for compensation of random delays.

The control algorithms were implemented using TDL. TDL is a domain-specific timing
language used for deterministic execution of control tasks. The TDL compiler
generates timing code that ensures the timely execution of the tasks. The real-time
control decision algorithms can be coded using Java, C or as Matlab functions. TDL
supports the notion of component modelling and segregates the task functionally details
from the timing aspects. TDL specifies the timing characteristics within a TDL module.
TDL module binds the task functions i.e. implementation of control laws, data
acquisition through sensors and actuators, into a timing module. Each TDL module acts
as a software component. The component structure enhances the maintainability and
reusability of the entire system. The details of TDL implementation and
TDLVisualCreator/Simulink modelling are explained in Appendix ‘A’.

The algorithms that use delay compensation are explained and validated successfully.
The predictive control algorithms are optimum for compensating the delay patterns
originating from deterministic communication networks. On the other hand, the
Adaptive Self-tuning regulator is applicable for compensating the delay patterns
originating from Non-Deterministic communication networks such as Ethernet. This
research lays the base for future extension of this project towards compensating the
unknown delays using neuro-fuzzy controllers.
6.2 Future Research

During the course of this research, future work areas were identified. The work described here incorporates and assumes network delay modelling using stochastic techniques. One future direction is to compensate for network delays with unpredictable and unknown behaviour. The compensation can be achieved using neuro-fuzzy controllers. Fuzzy control methods represent a new approach to the problems of controlling complex systems the systems whose mathematical model is difficult to describe or the systems with unpredictable disturbances.

The other area where this research can be refined and extended is the modelling and analysis of the network traffic patterns. The communication networks taken into consideration in this thesis Profibus DP and Gigabit Ethernet. The modelling principles described here can be applied to radio networks and wireless links used in hard real-time systems, specifically smart grid meters or robotics.

SCADA is the commonly used control data repository in many industrial processes. SCADA systems use the DNP3 application protocol for their communication channels. Delay Modelling for DNP3 can also serve to be a useful tool for optimizing the performance of SCADA system and hence industrial processes.

6.3 Publications

The project has been documented by 5 International Publications in Journals and Conferences. The publications are listed as follows:
Journal Publications


Conference Publications


APPENDICES

Appendix ‘A’: Time Definition Language (TDL) & TDLVisualCreator: A Tutorial

Appendix ‘B’: DataSocket Transport Protocol: An Introduction

Appendix ‘C’: Data Table for Inter-Arrival Time Gaps

1 Workstation Model
Appendix ‘A’

Time Definition Language (TDL) & TDLVisualCreator: A Tutorial

A.1 Time Definition Language

The Time Definition Language (TDL) is a domain-specific high-level timing language enabling creation of deterministic hard real-time control applications [1]. The TDL compiler automatically generates the timing code that ensures the specified behaviour on a given platform. TDL is integrated with SIMULINK through TDLVisualCreator. This Appendix will act as a guide for the code development and generation in SIMULINK using TDL VisualCreator tool.

Matlab/SIMULINK uses TDL library for generating the TDL modules.

Figure A.1 TDL Library in SIMULINK
As shown in Figure A.1, the TDL library is used to construct the TDL modules. The module is inserted into a new model file in SIMULINK. After inserting the module, TDL VisualCreator is loaded by double clicking the TDL module.

![Figure A.2 TDL VisualCreator](image)

Figure A.2 illustrates the TDL VisualCreator tool, where the module sensor, actuator and task definitions are implemented. This tutorial will consider the control problem for the second-order water tank process. The TDL/SIMULINK model constructed for the control of the process, in the presence of the network delays, is shown in Figure A.3.
The model constructs three TDL modules (shown greyed out in the centre of the Figure), that implement the PID controllers tuned for varying network delays. The task definitions in TDLVisualCreator are shown in Figures A.4.

As is evident in the figure A.4, the worst case execution time (WCET) is defined to be 20 milliseconds for the task. Figure A.5 depicts the period specified for the control task along with the automated TDL code.
Similarly, the control tasks are defined for other two modules. Each module specifies a different PID tuning for the delay.

**A.2 E-Machine & TDL Compiler**

![TDL Task Execution Environment](image)

**Figure A.6 TDL Task Execution Environment [1]**
TDL aims to control the execution of periodic tasks. Consider a small control program that reads a controller signal, updates the state and computes the output such as that shown in Figure A.8. Assume that the worst case execution time for a task is 10ms. TDL will assume the execution of a task according to logical execution time will exclude the reading and writing cycle time. Consequently the TDL task has to complete execution in 8ms i.e. execution time cycle.

![Figure A.7 TDL Mode Structure](image)

**Figure A.7 TDL Mode Structure**

TDL has made a successful attempt to simulate the control environment behaviour and provide higher degrees of predictability and compositionality, as compared to CORBA standards. Moreover TDL is only a specification language; it needs high level compiler support to implement the task functionality.
A.3 References


Appendix ‘B’

DataSocket Transport Protocol (DSTP):

An Overview

DataSocket Transport Protocol (DSTP) is an application layer protocol that works over TCP and acts as a data repository for time varying sensor data. It is National Instrument’s proprietary protocol that is used for transferring data to and from DataSocket Server Manager. This thesis presents DSTP traffic analysis and modelling over Ethernet.

DSTP is a connection-oriented protocol that utilizes the publish/subscribe pattern for data transfer. The server keeps track of client connection information. The clients that provide the measurement data are referred to as publishers and clients that consume the measurement data are referred to as subscribers. Publisher acquires the data from local or remote system and transfers it to the server. The subscribers connect to the server to read the published data. In a control scenario both controller and plant will need to subscribe and publish. The disadvantage in this case is that the subscribers can have access to the entire published data. This causes lot of overhead with redundant packets flowing over the network.

The DSTP traffic characteristics are analysed in this thesis. The response times for the data packets are recorded over Gigabit Ethernet. The inter-arrival time gaps for response packets are recorded for one subscriber, three subscribers and five
subscribers. The Table in Appendix ‘C’, records the inter-arrival time gaps for one subscriber.

**B.1 Data Transfer API Designed in Java**

DataSocket Server Manager acts as data repository for measurement data. Users interact with DSTP using THE DataSocket Application Programming Interface (API). The API has bindings for LabView, LAB Windows/CVI, Visual Basic and Visual C++. Since the control algorithms in this thesis are designed using Java platform, it was essential to create bindings for the DataSocket API in Java. The DataSocket API in Lab Windows/CVI was used and accessed from Java using the Java Native Interface (JNI).

The DSTP API comprises of:

1) A uniform resource locator (URL) for the communication protocol identifying the data item to be accessed. For DSTP, it is specified as

```
dstp://servername:port/dataitem
```

where `dataitem` is the named tag for the data and `servername` is the host name.

2) The DataSocket Reader for accessing the data item.

3) The DataSocket Writer for writing the value of the data item.

The API defined in LabWindows/CVI is used and called as native methods in Java. Java Native Interface (JNI) is a programming framework that allows the use of methods defined in other programming languages like C or Assembly to be invoked.
and used in Java. The methods are referred as native methods. In the JNI framework, native methods are implemented in separate .c or .cpp files. When the JVM invokes the function, it passes a JNIEnv pointer, a jobject pointer, and any Java arguments declared by the Java method. A JNI function may look like this:

```c
JNICALL JNIEXPORT void JNICALL Java_ClassName_MethodName
    (JNIEnv *env, jobject obj)
{
    /* Native Method */
}
```

The env pointer is a structure that contains the interface to the JVM. It includes all of the methods necessary to interact with the JVM and to work with Java objects.

To develop the DSTP interaction framework, two classes are defined that invoke the DataSocket Reader and Writer methods defined in LabWindows/CVI. The classes are:

**Listing B.1**

```java
public class DataSocketWriter
{
    public native void Connect();
    public native void Write();
    public native void Disconnect()
    static
    {
        System.loadLibrary("DataSocketWriter");
    }
}

public class DataSocketReader
{
    public native void Connect();
    public native void Read();
    public native void Disconnect()
    static
    {
        System.loadLibrary("DataSocketReader");
    }
}
```
Each class defines three native methods for:

a) Establishing the connection

b) Reading the data

c) Writing the data.

These native method definitions are given in the Listing B.2 and B.3. Before providing the listings, a brief detail is given about the DSTP native functions defined in LabWindows/CVI. The LabWindows/CVI API for DSTP connection includes three functions defined as:

1) `DS_OpenEx()`: This function is used to open the connection with a DataSocket Server Manager. The function prototype is given as:

```c
HRESULT DS_OpenEx (const char *URL, DSEnum_AccessModes accessMode, DSCallbackPtr eventFunction, void *callbackData, DSEnum_ExecutionMode executionMode, int timeout, DSHandle *DSHandle)
```

The parameters to this function are explained below:

a) `URL`: Defines the data source and data item to be read or written from the chosen DataSocket Server Manager.

b) `accessMode`: This parameter determines the type of access i.e. whether to read or write the DataSocket object. The access mode also determines when data is transferred from the data source to the DataSocket object or from the DataSocket object to the data source.

c) `eventFunction`: The callback handler function for the DataSocket object. The DataSocket object calls this callback when one or more of the following items have changed: the DataSocket object's data value, any of the DataSocket object's attributes, or the DataSocket object's status.
d) callbackData: The value that the DataSocket object will pass to your callback as the **callbackData** parameter. This can provide a unique identifier for each of several concurrent connections.

e) executionModel: This parameter determines the execution phase for the DataSocket object. It can be either Event based or Polling execution phase.

f) Timeout: This parameter specifies the number of milliseconds the DataSocket object waits for the connection to be established and after the timeout period has elapsed.

g) Status: This is the return value. The negative value indicates the function failure.

2) **DS_GetDataValue()**: This function is used to read a data item from DataSocket Server Manager. The function prototype is given as shown below:

```c
HRESULT DS_GetDataValue (DSHandle DSHandle, unsigned int type, void *value, unsigned int size, unsigned int *dimension1, unsigned int *dimension2);
```

Several of the parameters to this function are explained as:

a) DSHandle: This is the handle defined for DataSocket Object to identify the connection established using DS_OpenEx().

b) type: This parameter defines the type of the data item to be read. The following table lists the valid type constants and their corresponding C/Windows types:
Constant C/Windows type
CAVT_DOUBLE double
CAVT_FLOAT float
CAVT_LONG long
CAVT_SHORT short
CAVT_UCHAR unsigned char
CAVT_CSTRING char *
CAVT_BOOL short
CAVT_VARIANT VARIANT void *

c) size: This parameter defines the size of the buffer used to store the value of the data item.

d) Status: This is the return value. The negative value indicates the function failure.

3) DS_SetDataValue(): This function is used to write the value of the data item to the DataSocket source. The function prototype is defined as shown below:

HRESULT DS_SetDataValue (DSHandle DSHandle, unsigned int type, const void *value, unsigned int dimension1, unsigned int dimension2);

The parameters to this function are explained as:

a) DSHandle: This is the handle defined for DataSocket Object to identify the connection established using DS_OpenEx().

b) type: This parameter defines the type of the data item to be read. The following table lists the valid type constants and their corresponding C/Windows types:
Constant C/Windows type
CAVT_DOUBLE double
CAVT_FLOAT float
CAVT_LONG long
CAVT_SHORT short
CAVT_UCHAR unsigned char
CAVT_CSTRING char *
CAVT_BOOL short
CAVT_VARIANT VARIANT void *

c) value: This parameter defines the address of a variable that contains the data value.

The functions are invoked using Java Native Interface (JNI) calls. The JNI classes implemented are shown in Listings B.2 and B.3.

**Listing B.2**

// This listing implements the tasks defined in Class DataSocketReader
// Header Definitions
#include<jni.h>
#include<dataskt.h>
#include<cvidef.h>
#include "DataSocketReader.h"
// Handle Declaration
static DSHandle dsh;
double value;
// Function Prototype Definition
void CVICALLBACK DSCallback (DSHandle dsh, int event, void *calldata);

//JNI Function for Connecting to the DSTP Data Source
JNIEXPORT void JNIEXPORT Java_DataSocketReader_Connect(JNIEnv *env,
                                              jobject obj)
{
    HRESULT stat = S_OK;
    int isconnect;
    if(dsh)
    {
        stat = DS_DiscardObjHanlde (dsh);
        dsh = 0;
        printf(“Connection Closed”);
    }
    stat = DS_OpenEx(“dstp://localhost/ES2_ES2_009”,
                      DSConst_ReadAutoUpdate, DSCallback, NULL,
                      DS_EventModel, DSConst_Asynchronous, &dsh);
    printf(“Status of Connection %ld”, stat);
    return;
}
// JNI Function for Reading the Data Item from Data Source
JNIGLOBAL void JNICALL Java_DataSocketReader_Read(JNIEnv *env,
        jobject obj)
{
    HRESULT stat;
    stat = DS_GetDataValue(dsh, CAVT_DOUBLE, &value, sizeof(double), 0,
                           0);
    printf("Value fetched \%f\", value);
    return;
}

// JNI Function to Disconnect from the Data Source
JNIGLOBAL void JNICALL Java_DataSocketReader_DISCONNECT(JNIEnv *env,
                                                   jobject obj)
{
    HRESULT stat;
    if(dsh)
    {
        stat = DS_DsicardObjHandle (dsh);
        printf("Connection Closed\n");
        dsh = 0;
    }
    if(dsh == 0)
    {
        exit(EXIT_SUCCESS);
    }
}

void CVICALLBACK DSCallBack (DSHandle dsh, int event, void *calldata)
{
    HRESULT stat = S_OK;
    char message[1000];
    switch (event)
    {
    case DS_EVENT_STATUSUPDATED:
        stat = DS_GetLastMessage(dsh, message, 1000);
        printf("%s\", message);
        break;
    }
    return;
}

Listing B.3

// This listing implements the tasks defined in Class DataSocketWriter
// Header Definitions
#include<jni.h>
#include<dataskt.h>
#include<cvidef.h>
#include "DataSocketWriter.h"

// Handle Declaration
static DSHandle dsh;
double value;

// Function Prototype Definition
void CVICALLBACK DSCallback (DSHandle dsh, int event, void *calldata);

// JNI Function for Connecting to the DSTP Data Source
JNIGLOBAL void JNICALL Java_DataSocketWriter_CONNECT(JNIEnv *env,
jobject obj)
{
    HRESULT stat = S_OK;
    int isconnect;
    if(dsh)
    {
        stat = DS_DiscardObjHandle (dsh);
        dsh = 0;
        printf("Connection Closed");
    }
    stat = DS_OpenEx("dstp://localhost/ES2_ES2_009",
                    DSConst_ReadAutoUpdate, DSCallback, NULL,
                    DS_EventModel, DSConst_Asyncronous, &dsh);
    printf("status of connection %ld", stat);
    return;
}

//JNI Function for Writing the value of the Data Item to the Data Source
JNIEXPORT void JNICALL Java_DataSocketWriter_write(JNIEnv *env,
                                              jobject obj)
{
    HRESULT stat;
    stat = DS_SetDataVlaue(dsh, CAVT_DOUBLE, &value, Sizeof(double), 0,
                            0);
    printf("Value written %f", value);
    return;
}

// JNI Function to Disconnect from the Data Source
JNIEXPORT void JNICALL Java_DataSocketWriter_Dsiconnect(JNIEnv *env,
                                                      jobject obj)
{
    HRESULT stat;
    if(dsh)
    {
        stat = DS_DsicardObjHandle (dsh);
        printf("Connection Closed");
        dsh = 0;
    }
    if(dsh == 0)
    {
        exit(EXIT_SUCCESS);
    }
}

void CVICALLBACK DSCallBack (DSHandle dsh, int event, void *calldata)
{
    HRESULT stat = S_OK;
    char message[1000];
    switch (event)
    {
    case DS_EVENT_STATUSUPDATED:
        stat = DS_GetLastMessage(dsh, message, 1000);
        printf("%s", message);
        break;
    }
    return;
}
The LabWindows/CVI functions that are redefined using JNI calls are implemented in the GUI classes, defined in Java for establishing the connection with DSTP. The two main classes are designed using Java Swing Interface. The GUI interface is for the Connection with DSTP is shown in Figure B.1.

![Figure B.1 GUI Interface for DSTP Connection](image)

The GUI framework accepts the connection from NI’s DataSocket Server Manager and reads and writes the control data from and to the server.
Appendix ‘C’

Inter-Arrival Gaps for 1 Workstation over DSTP/Ethernet

<table>
<thead>
<tr>
<th>Frame No</th>
<th>Source IP Address</th>
<th>Destination IP Address</th>
<th>Arrival Time (in Seconds)</th>
<th>Time Delta from Pre Frame (in Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.000025</td>
<td>0.000021</td>
</tr>
<tr>
<td>2</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.000035</td>
<td>0.00001</td>
</tr>
<tr>
<td>3</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.000044</td>
<td>0.000009</td>
</tr>
<tr>
<td>4</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.00006</td>
<td>0.00001</td>
</tr>
<tr>
<td>5</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.000073</td>
<td>0.000009</td>
</tr>
<tr>
<td>6</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.000082</td>
<td>0.000009</td>
</tr>
<tr>
<td>7</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.000091</td>
<td>0.000009</td>
</tr>
<tr>
<td>8</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.0001</td>
<td>0.00001</td>
</tr>
<tr>
<td>9</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.000115</td>
<td>0.000009</td>
</tr>
<tr>
<td>10</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.000128</td>
<td>0.000023</td>
</tr>
<tr>
<td>11</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.000139</td>
<td>0.000009</td>
</tr>
<tr>
<td>12</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.000148</td>
<td>0.000009</td>
</tr>
<tr>
<td>13</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.000155</td>
<td>0.000009</td>
</tr>
<tr>
<td>14</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.010099</td>
<td>0.007955</td>
</tr>
<tr>
<td>15</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.155633</td>
<td>0.145534</td>
</tr>
<tr>
<td>16</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.155677</td>
<td>4.4E-05</td>
</tr>
<tr>
<td>17</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.155703</td>
<td>2.6E-05</td>
</tr>
<tr>
<td>18</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.155728</td>
<td>2.5E-05</td>
</tr>
<tr>
<td>19</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.155738</td>
<td>1E-05</td>
</tr>
<tr>
<td>20</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.155747</td>
<td>9E-06</td>
</tr>
<tr>
<td>21</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.155756</td>
<td>1.3E-05</td>
</tr>
<tr>
<td>22</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.155771</td>
<td>1.1E-05</td>
</tr>
<tr>
<td>23</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.155779</td>
<td>8E-06</td>
</tr>
<tr>
<td>24</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.155806</td>
<td>2.7E-05</td>
</tr>
<tr>
<td>25</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.155822</td>
<td>1.6E-05</td>
</tr>
<tr>
<td>26</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.155831</td>
<td>9E-06</td>
</tr>
<tr>
<td>27</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.155854</td>
<td>2.3E-05</td>
</tr>
<tr>
<td>28</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.155864</td>
<td>1E-05</td>
</tr>
<tr>
<td>29</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.155872</td>
<td>8E-06</td>
</tr>
<tr>
<td>30</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.166353</td>
<td>0.00856</td>
</tr>
<tr>
<td>31</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.274261</td>
<td>0.107908</td>
</tr>
<tr>
<td>32</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.302788</td>
<td>0.028527</td>
</tr>
<tr>
<td>33</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.30281</td>
<td>2.2E-05</td>
</tr>
<tr>
<td>34</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.30282</td>
<td>1E-05</td>
</tr>
<tr>
<td>35</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.302829</td>
<td>9E-06</td>
</tr>
<tr>
<td>36</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.302838</td>
<td>9E-06</td>
</tr>
<tr>
<td>37</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.302838</td>
<td>9E-06</td>
</tr>
<tr>
<td></td>
<td>IP Address</td>
<td>Timestamp</td>
<td>Data Value 1</td>
<td>Data Value 2</td>
</tr>
<tr>
<td>---</td>
<td>------------------</td>
<td>--------------------</td>
<td>--------------</td>
<td>---------------</td>
</tr>
<tr>
<td>38</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.302847</td>
<td>9E-06</td>
</tr>
<tr>
<td>39</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.302858</td>
<td>1.1E-05</td>
</tr>
<tr>
<td>40</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.302866</td>
<td>8E-06</td>
</tr>
<tr>
<td>41</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.302876</td>
<td>1E-05</td>
</tr>
<tr>
<td>42</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.302884</td>
<td>8E-06</td>
</tr>
<tr>
<td>43</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.302893</td>
<td>9E-06</td>
</tr>
<tr>
<td>44</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.302903</td>
<td>1E-05</td>
</tr>
<tr>
<td>45</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.302919</td>
<td>1.6E-05</td>
</tr>
<tr>
<td>46</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.302932</td>
<td>1.3E-05</td>
</tr>
<tr>
<td>47</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.302942</td>
<td>1E-05</td>
</tr>
<tr>
<td>48</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.312835</td>
<td>0.007909</td>
</tr>
<tr>
<td>49</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.449961</td>
<td>0.137126</td>
</tr>
<tr>
<td>50</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.449982</td>
<td>2.1E-05</td>
</tr>
<tr>
<td>51</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.449991</td>
<td>9E-06</td>
</tr>
<tr>
<td>52</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.45001</td>
<td>9E-06</td>
</tr>
<tr>
<td>53</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.450018</td>
<td>8E-06</td>
</tr>
<tr>
<td>54</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.450028</td>
<td>1E-05</td>
</tr>
<tr>
<td>55</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.450039</td>
<td>1.1E-05</td>
</tr>
<tr>
<td>56</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.450049</td>
<td>9E-06</td>
</tr>
<tr>
<td>57</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.450058</td>
<td>9E-06</td>
</tr>
<tr>
<td>58</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.450067</td>
<td>9E-06</td>
</tr>
<tr>
<td>59</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.450076</td>
<td>9E-06</td>
</tr>
<tr>
<td>60</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.450085</td>
<td>9E-06</td>
</tr>
<tr>
<td>61</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.450094</td>
<td>9E-06</td>
</tr>
<tr>
<td>62</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.450103</td>
<td>9E-06</td>
</tr>
<tr>
<td>63</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.460297</td>
<td>0.008195</td>
</tr>
<tr>
<td>64</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.575976</td>
<td>0.115679</td>
</tr>
<tr>
<td>65</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.600359</td>
<td>0.024383</td>
</tr>
<tr>
<td>66</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.600375</td>
<td>1.6E-05</td>
</tr>
<tr>
<td>67</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.600385</td>
<td>1E-05</td>
</tr>
<tr>
<td>68</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.600394</td>
<td>9E-06</td>
</tr>
<tr>
<td>69</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.600402</td>
<td>8E-06</td>
</tr>
<tr>
<td>70</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.600411</td>
<td>9E-06</td>
</tr>
<tr>
<td>71</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.600421</td>
<td>1E-05</td>
</tr>
<tr>
<td>72</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.600431</td>
<td>1E-05</td>
</tr>
<tr>
<td>73</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.600444</td>
<td>9E-06</td>
</tr>
<tr>
<td>74</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.600451</td>
<td>1.1E-05</td>
</tr>
<tr>
<td>75</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.600469</td>
<td>9E-06</td>
</tr>
<tr>
<td>76</td>
<td>134.115.133.132</td>
<td>134.115.129.88</td>
<td>0.600478</td>
<td>9E-06</td>
</tr>
<tr>
<td>77</td>
<td>134.115.133.132</td>
<td>134.115.129.88</td>
<td>0.600487</td>
<td>9E-06</td>
</tr>
<tr>
<td>78</td>
<td>134.115.133.132</td>
<td>134.115.129.88</td>
<td>0.600496</td>
<td>9E-06</td>
</tr>
<tr>
<td>79</td>
<td>134.115.133.132</td>
<td>134.115.129.88</td>
<td>0.610683</td>
<td>0.008188</td>
</tr>
<tr>
<td></td>
<td>IP Address</td>
<td>Time</td>
<td>Duration</td>
<td>Result</td>
</tr>
<tr>
<td>---</td>
<td>------------------</td>
<td>-------------------</td>
<td>----------</td>
<td>------------</td>
</tr>
<tr>
<td>82</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.755483</td>
<td>0.1448</td>
</tr>
<tr>
<td>83</td>
<td>134.115.133.132</td>
<td>134.115.129.88</td>
<td>0.755509</td>
<td>2.6E-05</td>
</tr>
<tr>
<td>84</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.755518</td>
<td>9E-06</td>
</tr>
<tr>
<td>85</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.755527</td>
<td>9E-06</td>
</tr>
<tr>
<td>86</td>
<td>134.115.133.132</td>
<td>134.115.129.88</td>
<td>0.755536</td>
<td>9E-06</td>
</tr>
<tr>
<td>87</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.755545</td>
<td>9E-06</td>
</tr>
<tr>
<td>88</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.755554</td>
<td>9E-06</td>
</tr>
<tr>
<td>89</td>
<td>134.115.133.132</td>
<td>134.115.129.88</td>
<td>0.755563</td>
<td>9E-06</td>
</tr>
<tr>
<td>90</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.755571</td>
<td>8E-06</td>
</tr>
<tr>
<td>91</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.75558</td>
<td>9E-06</td>
</tr>
<tr>
<td>92</td>
<td>134.115.133.132</td>
<td>134.115.129.88</td>
<td>0.755589</td>
<td>9E-06</td>
</tr>
<tr>
<td>93</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.755597</td>
<td>8E-06</td>
</tr>
<tr>
<td>94</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.755606</td>
<td>9E-06</td>
</tr>
<tr>
<td>95</td>
<td>134.115.133.132</td>
<td>134.115.129.88</td>
<td>0.755614</td>
<td>8E-06</td>
</tr>
<tr>
<td>96</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.755623</td>
<td>9E-06</td>
</tr>
<tr>
<td>97</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.765995</td>
<td>0.008366</td>
</tr>
<tr>
<td>98</td>
<td>134.115.133.132</td>
<td>134.115.129.88</td>
<td>0.90695</td>
<td>0.029184</td>
</tr>
<tr>
<td>99</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.906966</td>
<td>1.6E-05</td>
</tr>
<tr>
<td>100</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.906976</td>
<td>1E-05</td>
</tr>
<tr>
<td>101</td>
<td>134.115.133.132</td>
<td>134.115.129.88</td>
<td>0.906985</td>
<td>9E-06</td>
</tr>
<tr>
<td>102</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.907002</td>
<td>1.7E-05</td>
</tr>
<tr>
<td>103</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.907012</td>
<td>1E-05</td>
</tr>
<tr>
<td>104</td>
<td>134.115.133.132</td>
<td>134.115.129.88</td>
<td>0.907022</td>
<td>1E-05</td>
</tr>
<tr>
<td>105</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.907031</td>
<td>9E-06</td>
</tr>
<tr>
<td>106</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.90704</td>
<td>9E-06</td>
</tr>
<tr>
<td>107</td>
<td>134.115.133.132</td>
<td>134.115.129.88</td>
<td>0.907048</td>
<td>8E-06</td>
</tr>
<tr>
<td>108</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.907058</td>
<td>1E-05</td>
</tr>
<tr>
<td>109</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.907068</td>
<td>1E-05</td>
</tr>
<tr>
<td>110</td>
<td>134.115.133.132</td>
<td>134.115.129.88</td>
<td>0.907082</td>
<td>1.4E-05</td>
</tr>
<tr>
<td>111</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.907092</td>
<td>1E-05</td>
</tr>
<tr>
<td>112</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.9071</td>
<td>8E-06</td>
</tr>
<tr>
<td>113</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>0.917326</td>
<td>0.00824</td>
</tr>
<tr>
<td>114</td>
<td>134.115.133.132</td>
<td>134.115.129.88</td>
<td>1.057295</td>
<td>0.139969</td>
</tr>
<tr>
<td>115</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.057313</td>
<td>1.8E-05</td>
</tr>
<tr>
<td>116</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.057322</td>
<td>9E-06</td>
</tr>
<tr>
<td>117</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.057331</td>
<td>9E-06</td>
</tr>
<tr>
<td>118</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.05734</td>
<td>9E-06</td>
</tr>
<tr>
<td>119</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.057349</td>
<td>9E-06</td>
</tr>
<tr>
<td>120</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.057358</td>
<td>9E-06</td>
</tr>
<tr>
<td>121</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.057367</td>
<td>9E-06</td>
</tr>
<tr>
<td>122</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.057376</td>
<td>9E-06</td>
</tr>
<tr>
<td>123</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.057385</td>
<td>9E-06</td>
</tr>
<tr>
<td>124</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.057394</td>
<td>9E-06</td>
</tr>
<tr>
<td>125</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.057403</td>
<td>9E-06</td>
</tr>
<tr>
<td></td>
<td>IP Address</td>
<td>Time</td>
<td>1E-06</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.057412</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.057421</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.05743</td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.067718</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.202781</td>
<td></td>
</tr>
<tr>
<td>131</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.202795</td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.202804</td>
<td></td>
</tr>
<tr>
<td>133</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.202813</td>
<td></td>
</tr>
<tr>
<td>134</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.202822</td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.202831</td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.20284</td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.202849</td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.202858</td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.202867</td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.202876</td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.202885</td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.202894</td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.202902</td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.202911</td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.213222</td>
<td></td>
</tr>
<tr>
<td>146</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.344995</td>
<td></td>
</tr>
<tr>
<td>147</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.345016</td>
<td></td>
</tr>
<tr>
<td>148</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.345026</td>
<td></td>
</tr>
<tr>
<td>149</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.345035</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.345063</td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.345098</td>
<td></td>
</tr>
<tr>
<td>152</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.345107</td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.345116</td>
<td></td>
</tr>
<tr>
<td>154</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.345125</td>
<td></td>
</tr>
<tr>
<td>155</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.345139</td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.345149</td>
<td></td>
</tr>
<tr>
<td>157</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.345158</td>
<td></td>
</tr>
<tr>
<td>158</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.345166</td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.355799</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.494613</td>
<td></td>
</tr>
<tr>
<td>161</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.49463</td>
<td></td>
</tr>
<tr>
<td>162</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.49464</td>
<td></td>
</tr>
<tr>
<td>163</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.494649</td>
<td></td>
</tr>
<tr>
<td>164</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.494658</td>
<td></td>
</tr>
<tr>
<td>165</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.494667</td>
<td></td>
</tr>
<tr>
<td>166</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.494676</td>
<td></td>
</tr>
<tr>
<td>167</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.494685</td>
<td></td>
</tr>
<tr>
<td>168</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.494694</td>
<td></td>
</tr>
<tr>
<td>169</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.494703</td>
<td></td>
</tr>
<tr>
<td></td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.494712</td>
<td>9E-06</td>
</tr>
<tr>
<td>---</td>
<td>----------------</td>
<td>----------------</td>
<td>--------------</td>
<td>-------</td>
</tr>
<tr>
<td>171</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.494722</td>
<td>1E-05</td>
</tr>
<tr>
<td>172</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.49473</td>
<td>8E-06</td>
</tr>
<tr>
<td>173</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.494739</td>
<td>9E-06</td>
</tr>
<tr>
<td>174</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.494747</td>
<td>8E-06</td>
</tr>
<tr>
<td>175</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.505214</td>
<td>0.008464</td>
</tr>
<tr>
<td>176</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.636923</td>
<td>1.31709</td>
</tr>
<tr>
<td>177</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.636938</td>
<td>1.5E-05</td>
</tr>
<tr>
<td>178</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.636947</td>
<td>9E-06</td>
</tr>
<tr>
<td>179</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.636956</td>
<td>9E-06</td>
</tr>
<tr>
<td>180</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.636965</td>
<td>9E-06</td>
</tr>
<tr>
<td>181</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.636974</td>
<td>9E-06</td>
</tr>
<tr>
<td>182</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.636984</td>
<td>1E-05</td>
</tr>
<tr>
<td>183</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.636993</td>
<td>9E-06</td>
</tr>
<tr>
<td>184</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.637002</td>
<td>9E-06</td>
</tr>
<tr>
<td>185</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.63701</td>
<td>8E-06</td>
</tr>
<tr>
<td>186</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.637019</td>
<td>9E-06</td>
</tr>
<tr>
<td>187</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.637028</td>
<td>9E-06</td>
</tr>
<tr>
<td>188</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.637043</td>
<td>1.5E-05</td>
</tr>
<tr>
<td>189</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.637051</td>
<td>8E-06</td>
</tr>
<tr>
<td>190</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.63706</td>
<td>9E-06</td>
</tr>
<tr>
<td>191</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.64779</td>
<td>0.00873</td>
</tr>
<tr>
<td>192</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.786182</td>
<td>0.003204</td>
</tr>
<tr>
<td>193</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.786197</td>
<td>1.5E-05</td>
</tr>
<tr>
<td>194</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.786207</td>
<td>1E-05</td>
</tr>
<tr>
<td>195</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.786216</td>
<td>9E-06</td>
</tr>
<tr>
<td>196</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.786225</td>
<td>9E-06</td>
</tr>
<tr>
<td>197</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.78623</td>
<td>8E-06</td>
</tr>
<tr>
<td>198</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.786243</td>
<td>1E-05</td>
</tr>
<tr>
<td>199</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.786253</td>
<td>1E-05</td>
</tr>
<tr>
<td>200</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.786262</td>
<td>9E-06</td>
</tr>
<tr>
<td>201</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.786271</td>
<td>9E-06</td>
</tr>
<tr>
<td>202</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.786279</td>
<td>8E-06</td>
</tr>
<tr>
<td>203</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.786289</td>
<td>1E-05</td>
</tr>
<tr>
<td>204</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.786297</td>
<td>8E-06</td>
</tr>
<tr>
<td>205</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.786306</td>
<td>9E-06</td>
</tr>
<tr>
<td>206</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.786314</td>
<td>8E-06</td>
</tr>
<tr>
<td>207</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.796229</td>
<td>0.007582</td>
</tr>
<tr>
<td>208</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.929679</td>
<td>0.13345</td>
</tr>
<tr>
<td>209</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.929725</td>
<td>4.6E-05</td>
</tr>
<tr>
<td>210</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.929734</td>
<td>9E-06</td>
</tr>
<tr>
<td>211</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.929744</td>
<td>1E-05</td>
</tr>
<tr>
<td>212</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.929752</td>
<td>8E-06</td>
</tr>
<tr>
<td>213</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>1.929761</td>
<td>9E-06</td>
</tr>
<tr>
<td>Line</td>
<td>IP Address</td>
<td>Time Stamp</td>
<td>Value</td>
<td>Precision</td>
</tr>
<tr>
<td>------</td>
<td>------------</td>
<td>------------</td>
<td>-------</td>
<td>-----------</td>
</tr>
<tr>
<td>258</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.372555</td>
<td>1E-05</td>
</tr>
<tr>
<td>259</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.372564</td>
<td>9E-06</td>
</tr>
<tr>
<td>260</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.372573</td>
<td>9E-06</td>
</tr>
<tr>
<td>261</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.372582</td>
<td>9E-06</td>
</tr>
<tr>
<td>262</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.372591</td>
<td>9E-06</td>
</tr>
<tr>
<td>263</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.3726</td>
<td>9E-06</td>
</tr>
<tr>
<td>264</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.37261</td>
<td>1E-05</td>
</tr>
<tr>
<td>265</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.372619</td>
<td>9E-06</td>
</tr>
<tr>
<td>266</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.372627</td>
<td>8E-06</td>
</tr>
<tr>
<td>267</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.372636</td>
<td>9E-06</td>
</tr>
<tr>
<td>268</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.372645</td>
<td>9E-06</td>
</tr>
<tr>
<td>269</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.372653</td>
<td>8E-06</td>
</tr>
<tr>
<td>270</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.383139</td>
<td>0.008482</td>
</tr>
<tr>
<td>271</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.516307</td>
<td>0.029241</td>
</tr>
<tr>
<td>272</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.516331</td>
<td>2.4E-05</td>
</tr>
<tr>
<td>273</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.516341</td>
<td>1E-05</td>
</tr>
<tr>
<td>274</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.51635</td>
<td>9E-06</td>
</tr>
<tr>
<td>275</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.516359</td>
<td>9E-06</td>
</tr>
<tr>
<td>276</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.516368</td>
<td>9E-06</td>
</tr>
<tr>
<td>277</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.516378</td>
<td>1E-05</td>
</tr>
<tr>
<td>278</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.516387</td>
<td>9E-06</td>
</tr>
<tr>
<td>279</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.516396</td>
<td>9E-06</td>
</tr>
<tr>
<td>280</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.516406</td>
<td>1E-05</td>
</tr>
<tr>
<td>281</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.516417</td>
<td>1.1E-05</td>
</tr>
<tr>
<td>282</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.516427</td>
<td>1E-05</td>
</tr>
<tr>
<td>283</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.516435</td>
<td>8E-06</td>
</tr>
<tr>
<td>284</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.516444</td>
<td>9E-06</td>
</tr>
<tr>
<td>285</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.516453</td>
<td>9E-06</td>
</tr>
<tr>
<td>286</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.526699</td>
<td>0.00792</td>
</tr>
<tr>
<td>287</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.66859</td>
<td>0.141891</td>
</tr>
<tr>
<td>288</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.668608</td>
<td>1.8E-05</td>
</tr>
<tr>
<td>289</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.668618</td>
<td>1E-05</td>
</tr>
<tr>
<td>290</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.668626</td>
<td>8E-06</td>
</tr>
<tr>
<td>291</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.668635</td>
<td>9E-06</td>
</tr>
<tr>
<td>292</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.668644</td>
<td>9E-06</td>
</tr>
<tr>
<td>293</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.668653</td>
<td>9E-06</td>
</tr>
<tr>
<td>294</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.668662</td>
<td>9E-06</td>
</tr>
<tr>
<td>295</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.668672</td>
<td>1E-05</td>
</tr>
<tr>
<td>296</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.668683</td>
<td>1.1E-05</td>
</tr>
<tr>
<td>297</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.668693</td>
<td>1E-05</td>
</tr>
<tr>
<td>298</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.668701</td>
<td>8E-06</td>
</tr>
<tr>
<td>299</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.66871</td>
<td>9E-06</td>
</tr>
<tr>
<td>300</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.668719</td>
<td>9E-06</td>
</tr>
<tr>
<td>301</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>2.668728</td>
<td>9E-06</td>
</tr>
<tr>
<td>346</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.1156</td>
<td>8E-06</td>
</tr>
<tr>
<td>347</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.11561</td>
<td>1E-05</td>
</tr>
<tr>
<td>348</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.115618</td>
<td>8E-06</td>
</tr>
<tr>
<td>349</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.126309</td>
<td>0.008345</td>
</tr>
<tr>
<td>350</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.267109</td>
<td>0.1408</td>
</tr>
<tr>
<td>351</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.267125</td>
<td>1.6E-05</td>
</tr>
<tr>
<td>352</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.267135</td>
<td>1E-05</td>
</tr>
<tr>
<td>353</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.267144</td>
<td>9E-06</td>
</tr>
<tr>
<td>354</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.267154</td>
<td>1E-05</td>
</tr>
<tr>
<td>355</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.267163</td>
<td>9E-06</td>
</tr>
<tr>
<td>356</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.267187</td>
<td>2.4E-05</td>
</tr>
<tr>
<td>357</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.267196</td>
<td>9E-06</td>
</tr>
<tr>
<td>358</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.267206</td>
<td>1E-05</td>
</tr>
<tr>
<td>359</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.267214</td>
<td>8E-06</td>
</tr>
<tr>
<td>360</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.267223</td>
<td>9E-06</td>
</tr>
<tr>
<td>361</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.267232</td>
<td>9E-06</td>
</tr>
<tr>
<td>362</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.267241</td>
<td>9E-06</td>
</tr>
<tr>
<td>363</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.26725</td>
<td>9E-06</td>
</tr>
<tr>
<td>364</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.267259</td>
<td>9E-06</td>
</tr>
<tr>
<td>365</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.27767</td>
<td>0.000094</td>
</tr>
<tr>
<td>366</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.414599</td>
<td>0.022238</td>
</tr>
<tr>
<td>367</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.414613</td>
<td>1.4E-05</td>
</tr>
<tr>
<td>368</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.414622</td>
<td>9E-06</td>
</tr>
<tr>
<td>369</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.414632</td>
<td>1E-05</td>
</tr>
<tr>
<td>370</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.41464</td>
<td>9E-06</td>
</tr>
<tr>
<td>371</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.414649</td>
<td>9E-06</td>
</tr>
<tr>
<td>372</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.414666</td>
<td>1.1E-05</td>
</tr>
<tr>
<td>373</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.414669</td>
<td>9E-06</td>
</tr>
<tr>
<td>374</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.414677</td>
<td>8E-06</td>
</tr>
<tr>
<td>375</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.414686</td>
<td>9E-06</td>
</tr>
<tr>
<td>376</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.414696</td>
<td>1E-05</td>
</tr>
<tr>
<td>377</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.414707</td>
<td>1.1E-05</td>
</tr>
<tr>
<td>378</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.414717</td>
<td>1E-05</td>
</tr>
<tr>
<td>379</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.414725</td>
<td>8E-06</td>
</tr>
<tr>
<td>380</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.414734</td>
<td>9E-06</td>
</tr>
<tr>
<td>381</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.425133</td>
<td>0.008393</td>
</tr>
<tr>
<td>382</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.557523</td>
<td>0.13239</td>
</tr>
<tr>
<td>383</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.557533</td>
<td>1E-05</td>
</tr>
<tr>
<td>384</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.557561</td>
<td>2.8E-05</td>
</tr>
<tr>
<td>385</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.55757</td>
<td>9E-06</td>
</tr>
<tr>
<td>386</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.557585</td>
<td>1.5E-05</td>
</tr>
<tr>
<td>387</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.557595</td>
<td>1E-05</td>
</tr>
<tr>
<td>388</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.557604</td>
<td>9E-06</td>
</tr>
<tr>
<td>389</td>
<td>134.115.133.132</td>
<td>134.115.129.93</td>
<td>3.557613</td>
<td>9E-06</td>
</tr>
</tbody>
</table>
Conceptualization of Temporal Specifications for Application to Distributed Real-Time Control Systems

School of Engineering and Energy, Murdoch University
Conceptualization of Temporal Specifications for Application to Distributed Real-Time Control Systems

434 134.115.133.132 134.115.129.93 3.993458 1E-05
435 134.115.133.132 134.115.129.93 3.993472 1.4E-05
436 134.115.133.132 134.115.129.93 3.993482 1E-05
437 134.115.133.132 134.115.129.93 3.993494 1.2E-05
438 134.115.133.132 134.115.129.93 3.993503 9E-06
439 134.115.133.132 134.115.129.93 3.993512 9E-06
440 134.115.133.132 134.115.129.93 3.993521 9E-06
441 134.115.133.132 134.115.129.93 3.993545 2.4E-05
442 134.115.133.132 134.115.129.93 3.993554 9E-06
443 134.115.133.132 134.115.129.93 4.004235 0.00870 2
444 134.115.133.132 134.115.129.93 4.148043 0.143808