CPU Load Control of LTE Radio Base Station

Examensarbete utfört i reglertechnik
vid Tekniska högskolan vid Linköpings universitet
av

Joachim Larsson

LiTH-ISY-EX–15/4912–SE
Linköping 2015
CPU Load Control of LTE Radio Base Station

Examensarbete utfört i reglerteknik
vid Tekniska högskolan vid Linköpings universitet
av

Joachim Larsson

LiTH-ISY-EX–15/4912–SE

Handledare: Gustav Lindmark
ISY, Linköpings universitet
Tomas Bornefall
Ericsson AB

Examinator: Martin Enqvist
ISY, Linköpings universitet

Linköping, 27 november 2015
A radio base station (RBS) may become overloaded if too many mobile devices communicate with it at the same time. This could happen at for instance sport events or in the case of accidents. To prevent CPU overload, the RBS is provided with a controller that adjusts the acceptance rate, the maximum number of connection requests that can be accepted per time interval. The current controller is tuned in real radio base stations and the procedure is both time consuming and expensive. This, combined with the fact that the mobile data usage is predicted to increase puts more pressure on today's system. Thus, there is a need to be able to simulate the system in order to suggest an alternative controller.

In this thesis, an implementation of the system is developed in MATLAB in order to simulate the RBS system load control behaviour. A CPU load model is estimated using system identification. The current version of the CPU load controller and an alternative PI CPU load controller are implemented. Both are evaluated on different test cases and this shows that it is possible to increase the performance of the system with the alternative CPU load controller, both in terms of lower amount of rejected connection requests and decreased CPU load overshoot.
Abstract

A radio base station (RBS) may become overloaded if too many mobile devices communicate with it at the same time. This could happen at for instance sport events or in the case of accidents. To prevent CPU overload, the RBS is provided with a controller that adjusts the acceptance rate, the maximum number of connection requests that can be accepted per time interval. The current controller is tuned in real radio base stations and the procedure is both time consuming and expensive. This, combined with the fact that the mobile data usage is predicted to increase puts more pressure on today’s system. Thus, there is a need to be able to simulate the system in order to suggest an alternative controller.

In this thesis, an implementation of the system is developed in MATLAB in order to simulate the RBS system load control behaviour. A CPU load model is estimated using system identification. The current version of the CPU load controller and an alternative PI CPU load controller are implemented. Both are evaluated on different test cases and this shows that it is possible to increase the performance of the system with the alternative CPU load controller, both in terms of lower amount of rejected connection requests and decreased CPU load overshoot.
Acknowledgments

This thesis has been performed at LMR SM Characteristics at Ericsson AB in Linköping. First of all I would like to thank my supervisor Tomas Bornefall for all the help and guidance throughout the thesis. He has showed great interest in the work and always found time for questions. I would also like to thank the rest of my colleagues at LMR SM Characteristics for making my time at Ericsson AB a pleasant experience.

Secondly I would like to thank my supervisor Gustav Lindmark and examiner Martin Enqvist at ISY for helpful advice during the thesis. You have always been available to discuss the thesis and guided me on the right path.

Finally, I would like to thank my family and friends for always supporting me and making my experience at Linköping University great.

Linköping, November 2015
Joachim Larsson
# Contents

Notation ix  

1 Introduction 1  
1.1 Background 1  
1.2 Problem Formulation 2  
1.3 Limitations 2  
1.4 Previous Work 3  
1.5 Thesis Outline 4  

2 Telecommunication 5  
2.1 Historical Overview 5  
2.2 Network Architecture 7  
2.3 LTE Radio Base Station 8  
2.4 LTE Radio Resource Control 10  

3 System Identification 15  
3.1 Overview 15  
3.2 The Prediction-Error Method 16  
3.3 Model Structures 16  
3.4 Model Validation 19  

4 System Overview and Modelling 21  
4.1 CPU Load Control System 21  
4.2 CPU Load Model 23  
4.2.1 Experimental Method and Data 23  
4.2.2 Modelling Results 24  
4.3 User Behaviour Model 28  
4.4 Discussion 32  

5 Control Design 37  
5.1 Current CPU Load Controller 37  
5.2 Alternative PI CPU Load Controller 39  
5.3 Evaluation 41
### Abbreviations and Acronyms in Automatic Control

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARMAX</td>
<td>Auto-Regressive Moving Average with eXogeneous input</td>
</tr>
<tr>
<td>ARX</td>
<td>Auto-Regressive with eXogeneous input</td>
</tr>
<tr>
<td>BJ</td>
<td>Box-Jenkins</td>
</tr>
<tr>
<td>FIR</td>
<td>Finite Impulse Response</td>
</tr>
<tr>
<td>NARX</td>
<td>Non-linear Auto-Regressive with eXogenous input</td>
</tr>
<tr>
<td>OE</td>
<td>Output Error</td>
</tr>
<tr>
<td>PEM</td>
<td>Prediction-Error Method</td>
</tr>
<tr>
<td>P</td>
<td>Proportional Controller</td>
</tr>
<tr>
<td>I</td>
<td>Integral Controller</td>
</tr>
<tr>
<td>PI</td>
<td>Proportional Integral Controller</td>
</tr>
<tr>
<td>PID</td>
<td>Proportional Integral Derivative Controller</td>
</tr>
<tr>
<td>STD</td>
<td>Standard Deviation</td>
</tr>
</tbody>
</table>
# Abbreviations and Acronyms in Telecommunication

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIAT</td>
<td>Burst Inter-Arrival Time</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>EDGE</td>
<td>Enhanced Data Rates for GSM Evolution</td>
</tr>
<tr>
<td>E-NODEB</td>
<td>Evolved NODEB</td>
</tr>
<tr>
<td>GPRS</td>
<td>General Packet Radio Service</td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile Communications</td>
</tr>
<tr>
<td>HD</td>
<td>High Definition</td>
</tr>
<tr>
<td>HSPA</td>
<td>High Speed Packet Access</td>
</tr>
<tr>
<td>IAT</td>
<td>Inactivity Time</td>
</tr>
<tr>
<td>LTE</td>
<td>Long Term Evolution</td>
</tr>
<tr>
<td>RBS</td>
<td>Radio Base Station</td>
</tr>
<tr>
<td>RRC</td>
<td>Radio Resource Control</td>
</tr>
<tr>
<td>RTOS</td>
<td>Real-Time Operating System</td>
</tr>
<tr>
<td>SIM</td>
<td>Subscriber Identity Module</td>
</tr>
<tr>
<td>SMS</td>
<td>Short Message Service</td>
</tr>
<tr>
<td>UE</td>
<td>User Equipment</td>
</tr>
<tr>
<td>UMTS</td>
<td>Universal Mobile Telecommunications System</td>
</tr>
<tr>
<td>VoLTE</td>
<td>Voice Over LTE</td>
</tr>
</tbody>
</table>
## Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>CPU load</td>
</tr>
<tr>
<td>$\bar{y}$</td>
<td>Moving average CPU load</td>
</tr>
<tr>
<td>$\hat{y}$</td>
<td>Simulated CPU load</td>
</tr>
<tr>
<td>$T_s$</td>
<td>Sample time of CPU load measurements and sample time in simulation</td>
</tr>
<tr>
<td>$a_r$</td>
<td>Acceptance rate, a limit of the maximum number of connection requests that can be accepted per time window</td>
</tr>
<tr>
<td>$a_{r,\text{max}}$</td>
<td>A limit of the maximum acceptance rate</td>
</tr>
<tr>
<td>$a_{r,\text{min}}$</td>
<td>A limit of the minimum acceptance rate</td>
</tr>
<tr>
<td>$W_L$</td>
<td>Length of time window</td>
</tr>
<tr>
<td>$S_{\text{req,W_L}}$</td>
<td>The cumulative number of connection requests per time window</td>
</tr>
<tr>
<td>$n_{\text{req,T_s}}$</td>
<td>The number of connection requests per CPU load measurement</td>
</tr>
<tr>
<td>$n_{\text{req,T_s}}$</td>
<td>The number of rejected connection requests per CPU load measurement</td>
</tr>
<tr>
<td>$n_{\text{acc,T_s}}$</td>
<td>The number of accepted connection requests per CPU load measurement</td>
</tr>
<tr>
<td>$c_1, c_2, c_3$</td>
<td>CPU load thresholds</td>
</tr>
<tr>
<td>$d_1, d_2$</td>
<td>Dataset 1 and 2 respectively</td>
</tr>
<tr>
<td>$r$</td>
<td>Reference CPU load</td>
</tr>
<tr>
<td>$\delta$</td>
<td>Constant step that the current controller adjusts $a_r$ with</td>
</tr>
<tr>
<td>$B_s$</td>
<td>Burst size</td>
</tr>
<tr>
<td>$K_p$</td>
<td>Proportional constant for the PI controller</td>
</tr>
<tr>
<td>$K_i$</td>
<td>Integrating constant for the PI controller</td>
</tr>
<tr>
<td>$F_f$</td>
<td>Fitness function to evaluate the performance of the controllers</td>
</tr>
<tr>
<td>$F_{L,\text{rej}}$</td>
<td>Linear rejection part of the $F_f$</td>
</tr>
<tr>
<td>$F_{Q,\text{over}}$</td>
<td>Quadratic overshoot part of the $F_f$</td>
</tr>
</tbody>
</table>
1.1 Background

Radio Base Stations (RBS) are the first network entities a mobile device communicates with in a mobile communication network. A connection request from a mobile device goes over an air interface to the RBS. It continues by cables through the mobile communication network to either another mobile device or the Internet. Lately, there has been an increasing usage of data in mobile devices. This has led to the development of the fourth generation of mobile communication networks (4G), also referred to as Long-Term Evolution (LTE). The RBS in LTE is called E-NODEB. An RBS may become overloaded if too many mobile devices communicates with it at the same time. This could happen at for instance sport events or accidents. Overload is a state when the CPU load is higher than its capacity and this could lead to delays for the end user, or in a worst case scenario, a full system reset.

To prevent CPU overload, the RBS is provided with an overload protection mechanism. This mechanism is a controller with the purpose to keep the CPU load below a defined maximum limit by rejecting incoming connection requests. The current controller implementation is ad hoc and tuned in real RBSs based on observations. The RBSs are connected to load generators that simulate load from mobile devices. The hardware is expensive and it is difficult and time consuming to try out a new controller. Therefore, there is a need to be able to simulate the RBS system load control behaviour in order to experiment with alternative controllers based on established control theory. Such a controller could improve the quality of service for the end user by reducing the number of rejected connection requests. It could also be possible to improve the performance by decreasing the CPU load overshoot and thereby not block resources intended for other activities.
1 Introduction

Figure 1.1: Simplified signal flow in an RBS. A mobile device sends a capacity request to the overload protection. The overload protection decides whether to accept or reject the request depending on the measured CPU load.

This thesis aims at developing tools for simulation of the CPU load in an RBS and suggesting an alternative controller for the system.

1.2 Problem Formulation

The simulation requires a model describing how the CPU load in LTE E-NODEB base stations varies with incoming connection requests, an overload protection controller that rejects incoming connection requests if the acceptance rate and thereby the CPU load is too high and a statistical user behaviour model that can simulate connection requests from mobile phones and how they react to accepts and rejects. See Figure 1.1 for a simplified illustration of the signal flow in an RBS. The following questions will be studied:

- What dynamic model is suitable for modeling of the CPU load in an RBS?
- How can a statistical user behaviour model be described?
- Is it possible to reproduce the RBS system load control behaviour in a simulated environment?
- What controller based on established control theory is suitable for overload protection in an RBS?
- Is it possible to improve the performance, in terms of lower amount of rejected connection requests and decreased CPU load overshoot with such a proposed controller?

1.3 Limitations

The RBS CPU has a multicore architecture. However, only the load of the highest loaded core is measured. Connection requests from mobile phones in different geographical regions are distributed to different cores and it is not possible to log which core a specific request is being processed on. The datasets have simulated UEs which are evenly distributed in the regions. Thereby, all cores should be evenly loaded.
The CPU load is affected not only by the accepted signals but also by the rejected ones and other signals. The datasets have a low number of rejects and this effect is therefore not included in the CPU model. The CPU load model takes the accepted connection request messages as input.

1.4 Previous Work

There is an extensive literature base about system identification in general and a thorough description is presented in [15]. An introduction to system identification and some practical aspects of it is described in [17]. Non-linear models and neural networks are described in [20]. A MATLAB approach can be found in [19].

There is a lot of research on load prediction of distributed systems. Host load prediction using linear models such as AR and ARMA is evaluated in [9]. They show that the host load is predictable from previous behaviour and choose AR for simplicity. A relatively high model order of 16 is used. Modelling of CPU load in distributed systems using adaptive network-based fuzzy interference systems is described in [5] and [6] where the authors claim that their approach performs better than exponential smoothing and AR. A multi-step-ahead prediction of CPU load is evaluated in [23]. Multi-step-ahead prediction is a more challenging task than one-step-ahead prediction.

The PID controller is a very common controller in the industry and an introduction to the subject is presented in [18]. For a more thorough description one could read [4].

The third generation of mobile telecommunication networks (3G) is described in [12]. A focused description of the evolution from UMTS (3G) to LTE (4G) is presented in [13]. For a thorough reading about LTE in general a start is [7, 8]. LTE radio resource control (RRC) is a protocol handling the control signalling, the signalling that affects the CPU load. The major functions of the RRC protocol are to establish connection and release between the mobile device and the RBS and this is described in detail in [14].
1.5 Thesis Outline

An introduction to telecommunication is given in Chapter 2 and starts with a historical overview of mobile telecommunication networks. It is followed by a more detailed description of LTE and the LTE RBS. The radio resource control protocol is used to establish connection and release between a UE and an RBS and a simplified version is presented. The report continues with a brief description of system identification in Chapter 3. It begins with the definition of an objective function that is minimized by optimization. Concepts and details of both linear and non-linear model structures are described. The chapter ends with details about model validation. In Chapter 4, the system, implemented in MATLAB and SIMULINK, is described. The proposed CPU load model is presented together with the user behaviour model describing a UE's behaviour in the mobile telecommunication network. The chapter ends with a discussion of the presented CPU load model and the limitations of it. Chapter 5 describes the current implementation of the overload protection as well as the proposed controller. The controllers are evaluated in different test cases and compared by defining a fitness function. The final Chapter 6 is a conclusion on what the implemented system in this thesis can be used for.
This chapter begins with a historical overview of mobile telecommunication networks and introduces the reader to telecommunication concepts and trends. It continues with a description of the network architecture and describes how a phone call is made. The chapter continues with a more detailed description of the RBS and its internal architecture. The chapter ends with a description of the LTE radio resource control (RRC) protocol which is used to establish connections and releases between a mobile device and an RBS. RRC is the signalling that affects the CPU load and thus the main focus of this thesis.

### 2.1 Historical Overview

In order to understand some concepts of LTE it is important to go back in history and see how the telecommunication network has evolved from previous generations. See Figure 2.1 for a time line of the development.

![Figure 2.1: Time of release for different generations of telecommunication networks.](image)

In the early 1980s, the first generation of wireless mobile communication technology (1G) was deployed [21]. It was based on analogue techniques and with today’s standard inefficient in terms of the use of the radio spectrum. The mobile devices were expensive and not a common consumer product.
The second generation of wireless mobile communication technology (2G), called GSM (Global System for Mobile Communication), arrived in the early 1990s [13]. It was based on digital technology and used the radio spectrum more efficiently compared to 1G. The mobile devices were smaller and a more common consumer product. A digital telecommunication network consists of a radio access network, a core network and authentication of mobile devices via SIM (Subscriber Identity Module) cards. The radio access network handles everything related to radio functionality such as radio-resource handling and retransmission protocols [8]. The core network on the other hand handles everything that is not related to radio functionality, such as authentication, charging and connection between two mobile devices. The core network consists of two types of communication, circuit switched and packet switched. In general, phone calls use circuit switched communication while packet switched communication is used for web pages, emails etc. [7]. At first the 2G network could only handle voice traffic but it was later extended with functionality for instant messaging, called SMS (Short Message Service). At this time, the Internet became widely used and with that the demand from the end users to download data to their mobile devices. This was introduced with GPRS (General Packet Radio Service) and EDGE (Enhanced Data Rates for GSM Evolution), which are often referred to as 2.5G and 2.75G, respectively. At the time of release, GPRS offered data rates of 14kbps in the uplink and 40kbps in the downlink [22]. Uplink is the transmission from the mobile device to the network and downlink is the reversed direction.

In the early 2000s the third generation of wireless mobile communication technology (3G), called UMTS (Universal Mobile Telecommunication System), was deployed and it offered data rates of 64kbps and 384kbps in the up- and downlink, respectively [12]. Functionality for video streaming and video calls was also introduced. UMTS was based on GSM but used new techniques. The technology in the air interface was changed but the core network was kept. In year 2005, HSPA (High-Speed Packet Access) was deployed due to the higher usage of data traffic. HSPA is often referred to as 3.5G and offered data rates of 5.8Mbps in the uplink and 13.4Mbps in the downlink [22].

The growth of mobile data usage led to the fourth generation of wireless mobile communication technology (4G), called LTE (Long-Term Evolution). At the time of release, LTE offered data rates of 75Mbps and 300Mbps in the up- and downlink, respectively [21] and the possibility to stream video in HD (High Definition) quality. The data usage is constantly increasing and puts more pressure on today’s system and for instance the overload protection. The predicted data usage at year 2020 can be seen in Figure 2.2. In order to meet the upcoming demand, 5G is under development [1].
2.2 Network Architecture

As previously mentioned, a telecommunication network consists of a radio access network, a core network and mobile devices. A mobile device could be a mobile phone, a tablet, a laptop or such, and will hereby be referred to as user equipment (UE). See Figure 2.3 for an illustration of a telecommunication network.

There are two types of communication used in a telecommunication network, circuit switched and packet switched. In circuit switched communication, phone calls in a geographic region are transported to the network operator who covers that region. The operator in turn communicates with the public switched telephone network to make calls with operators in other regions. In circuit switching, a dedicated two-way connection is established for each phone call. Because of this, data is transmitted at a constant data rate with minimal delay. In packet switching on the other hand, data is divided into packets. The packets are labelled with a destination address and forwarded to the destination via routers. The network resources are shared by all the users in the network which makes it more efficient than circuit switched communication. However, delays may occur if too many UEs transmit simultaneously [7]. LTE does not support circuit switched communication and thereby has to use the GSM or UMTS network for regular phone calls. A new technique called voice over LTE (VoLTE) was released in 2012 and made it possible to send voice calls as data streams over the LTE core network [3].

The radio access network in LTE consists of one or several radio base stations. Thus, LTE has a flat radio access network architecture [8]. This differs from the radio access network in GSM and UMTS which consists of a controller and an RBS.

**Figure 2.2:** The data usage each month is predicted to grow exponentially which puts more pressure on today’s systems. Ericsson predicts that the monthly data usage in the year 2020 will be 30.5 ExaBytes (1 ExaByte = \(10^6\) TeraByte = \(10^{18}\) Byte). Data from [2].
Figure 2.3: A UE sends a message over an air interface (dashed arrow) to the radio access network which consists of radio base stations. The message is forwarded to the core network through cables (solid arrows). Depending on the message, the core network forwards it to either the public switched telephone network for phone calls, or to a packet data network such as the Internet.

In LTE the controller is built into the RBS which allows faster response times.

2.3 LTE Radio Base Station

The e-NODEB is the RBS in LTE and is in control of all radio-related functions and several e-NODEB units are distributed throughout a network’s coverage area. The e-NODEB is also responsible for control signalling functions such as radio resource management, which handles allocation of resources based on connection requests and scheduling of traffic to achieve better quality of service. The e-NODEB is the upgraded version of NODEB, which is the UMTS radio base station. For simplicity, the e-NODEB will hereby be referred to as the RBS. The RBS consists of three major parts, the antennas, the radio units and the digital units. An antenna is used to receive and transmit radio signals from and to a UE over an air interface. A system often consists of three directed antennas, each covering a cell, which is a geographical 120° sector around the RBS. A radio unit modulates and demodulates all signals received by and transmitted from the antenna. A digital unit handles all digital signal processing and is a parallel processing system. It is also the interface to the core network. The radio unit and digital unit are located in a cabinet, for instance Ericsson RBS 6120 which can be seen in Figure 2.4. A cabinet also contains power supply and cooling. A digital unit has a maximum number of users that can be handled so a cabinet can hold several digital units. See Figure
Figure 2.4: Ericsson RBS 6120 cabinet with 6 digital units and 12 radio units. In this configuration, each row in the cabinet has 3 digital units to the left and 6 radio units to the right. Reproduced with permission from Ericsson AB.

2.5 for a picture of a digital unit.
A digital unit consists of the base band signal processing unit, a main processor and the overload protection. The user data is forwarded from the radio unit to the digital unit where it is processed in the signal base band. This does not influence the CPU load. The control signalling, that is the functionality to establish a new connection, is processed in the CPU and thereby not part of the base band signal processing. Thus, the control signalling influences the CPU load and is analysed in this thesis. The overload protection resides in the CPU and thus causes load itself. Thereby there is a restriction on the overload protection to be computationally efficient. The CPU is running a real-time operating system (RTOS). The RTOS is called soft in the way that it is not mandatory to meet the deadline every time for every task. See Figure 2.6 for an illustration of a digital unit.

2.4 LTE Radio Resource Control

LTE radio resource control (RRC) is a protocol handling the control signalling, that is, the signalling that affects the CPU load. The major functions of the RRC protocol are to establish connection and release between the UE and the RBS [14]. A UE can be in either idle or connected state relative to an RBS. Transition from one state to another is triggered by an event or a condition. See Table 2.1 for the two states of a UE. A simplified version of the RRC protocol is illustrated in Figure 2.7 and in Table 2.2. For a complete description of the RRC protocol see [14].
2.4 LTE Radio Resource Control

Figure 2.6: Internal structure of a digital unit and the flow of signals. When a UE transmits a signal, an antenna receives the signal and modulates and demodulates it in the radio unit. The signal is transmitted to the digital unit where it is processed in the signal base band. The user data is transmitted to the core network. The control signals are processed in the CPU where the overload protection resides. Only the control signals influence the CPU load.

Table 2.1: A UE is a state machine with two states, idle and connected.

<table>
<thead>
<tr>
<th>UE state</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>A UE that is not connected to an RBS.</td>
</tr>
<tr>
<td>Connected</td>
<td>A UE that is connected to an RBS and can transmit data.</td>
</tr>
</tbody>
</table>
Figure 2.7: A simplified version of the LTE RRC protocol. A UE in the idle state sends a connection request message to the RBS. If the RBS has capacity for the request, as in this example, the RBS sends a setup message to the UE. The UE completes the connection by sending an accept message. The UE is now in the connected state and is permitted to transmit data. If the UE is inactive for a certain period of time, it is released from the RBS and transitions to the idle state. The RRC messages are control signals that are processed on the main CPU, thus influencing the CPU load.
### Table 2.2: Messages in the LTE RRC protocol.

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>A UE in the idle state initiates a connection by sending a request message.</td>
</tr>
<tr>
<td>Reject</td>
<td>The overload protection in the RBS decides that there is not enough capacity in the RBS to accept the request. The RBS sends a reject message to the UE. The UE is forced to try to initiate a connection at a later time.</td>
</tr>
<tr>
<td>Setup</td>
<td>The overload protection in the RBS decides that there is enough capacity in the RBS to accept the request. The RBS sends a setup message to the UE.</td>
</tr>
<tr>
<td>Accept</td>
<td>A UE receives a setup message from the RBS and completes the connection procedure by sending a setup complete message. The UE now transitions from the idle state to the connected state and can now transmit data.</td>
</tr>
<tr>
<td>Release</td>
<td>A connected UE has been inactive too long, that is, has not transmitted data for a period of time, and is released from the RBS. The UE now transitions from the connected state to the idle state. A UE is released in order to save battery time on the UE and at the same time clear memory in the RBS.</td>
</tr>
</tbody>
</table>
This chapter begins with an overview of system identification, which is used to estimate the CPU load model. It continues with the prediction-error method which is one method that can be used to fit a model to data. Thereafter, linear model structures followed by non-linear model structures are explained. The chapter ends with the concept of model validation.

### 3.1 Overview

The idea of system identification is to build a dynamic model based on measurements from a system. In other words, to estimate the relationship between the input and the output of the system. First, a model structure is chosen. We continue by defining an objective function that we minimize by optimization. By validating the model on a new dataset we ensure that the model is good enough for our application. If the model is not sufficient for our application, we start over. That is, the system identification process is iterative. See Figure 3.1 for an illustration of a system. For a more thorough description of system identification see [15], [17], [11] and [16].

![Figure 3.1: Schematic of a dynamic system with input $u$, disturbance $v$ and output $y$.](image-url)
3.2 The Prediction-Error Method

The prediction-error method (PEM) is used to fit a model to data. The parameters are estimated by minimizing the loss function

$$V_N(\theta) = \frac{1}{N} \sum_{t=1}^{N} \ell(\varepsilon(t, \theta))$$

(3.1)

where $\ell$ is a scalar-valued function, $\varepsilon$ the prediction error and $\theta$ the parameter vector. A standard choice of $\ell$ is the quadratic function $\ell(\varepsilon) = \frac{1}{2} \varepsilon^2$. The prediction error $\varepsilon$ can be written

$$\varepsilon(t, \theta) = y(t) - \hat{y}(t|\theta)$$

(3.2)

where $\hat{y}(t|\theta)$ is the predicted output at time $t$. Thus, the problem is to find the parameter vector $\theta$ that gives the smallest prediction error in this sense according to

$$\hat{\theta}_N = \arg\min_{\theta} V_N(\theta).$$

(3.3)

3.3 Model Structures

A linear discrete time system with additive disturbance can be described as

$$y(t) = G(q)u(t) + v(t)$$

(3.4)

where $y(t)$ is the output, $u(t)$ the input, $v(t)$ the disturbance and $G(q)$ the transfer function between input and output. The shift operator $q$ is defined as $qu(t) = u(t + 1)$. A common way to describe the disturbance is

$$v(t) = H(q)e(t)$$

(3.5)

where $e(t)$ is white noise and $H(q)$ a filter. In order to handle the fact that the transfer functions $G(q)$ and $H(q)$ are unknown we introduce the parameter vector
\[ y(t) = G(q, \theta)u(t) + H(q, \theta)e(t). \]  
(3.6)

A common way to evaluate different models is by comparing their ability to predict the next output. By multiplying both sides with \( H^{-1}(q, \theta) \) and adding \( y(t) \) we can rewrite (3.6) as

\[ y(t) = (I - H^{-1}(q, \theta))y(t) + H^{-1}(q, \theta)G(q, \theta)u(t) + e(t). \]  
(3.7)

Thus, the one-step-ahead predictor of \( y(t) \) is

\[ \hat{y}(t|\theta) = (1 - H^{-1}(q, \theta))y(t) + H^{-1}(q, \theta)G(q, \theta)u(t). \]  
(3.8)

A way to categorize model structures is to color-code them in black, white and grey. The parameters of a black-box model have no physical meaning while the parameters of a white-box model are derived from physical laws. The grey-box model is in-between the black-box model and white-box model, and the parameters have some physical interpretation. For black-box models it is common to describe \( G(q, \theta) \) and \( H(q, \theta) \) as rational transfer functions in the shift operator. We thus get

\[ G(q, \theta) = \frac{B(q)}{F(q)} = \frac{b_1 q^{-n_k} + b_2 q^{-n_k-1} + \cdots + b_{n_k} q^{-n_k-n_b+1}}{1 + f_1 q^{-1} + \cdots + f_{n_f} q^{-n_f}} \]  
(3.9)

and

\[ H(q, \theta) = \frac{C(q)}{D(q)} = \frac{1 + c_1 q^{-1} + \cdots + c_{n_c} q^{-n_c}}{1 + d_1 q^{-1} + \cdots + d_{n_d} q^{-n_d}} \]  
(3.10)

where \( n_b, n_c, n_d, n_f \) are structural parameters and \( n_k \) is the time delay. The parameter vector \( \theta \) contains \( b_i, c_i, d_i \) and \( f_i \). Equation (3.6) together with \( G(q, \theta) \) and \( H(q, \theta) \) as in (3.9) and (3.10) is called a Box-Jenkins (BJ) model. See Figure 3.2 for schematic of the Box-Jenkins model.

There exist special cases of the general Box-Jenkins model structure. With \( H(q) = 1 \), that is, \( n_c = n_d = 0 \) we get the Output-Error (OE) model. With \( F(q) = D(q) = A(q) = 1 + a_1 q^{-1} + \cdots + a_{n_a} q^{-n_a} \) and by multiplying (3.6) by \( A(q) \) we get

\[ A(q)y(t) = B(q)u(t) + C(q)e(t) \]  
(3.11)

which is called the Auto-Regressive Moving Average with eXogeneous input (ARMAX) model. With \( C(q) = 1 \), that is \( n_c = 0 \) in (3.11), we get the ARX model.

The linear regression model structure can be written

\[ \hat{y}(t|\theta) = \varphi^T(t)\theta \]  
(3.12)

where \( \varphi(t) \) is the regression vector which in the ARX case is

\[ \varphi(t) = [-y(t-1), ..., -y(t-n_a), u(t-n_k), ..., u(t-n_k-n_b+1)]^T. \]  
(3.13)
and
\[
\theta = [a_1, ..., a_{n_a}, b_1, ..., b_{n_b}]^T. \tag{3.14}
\]

In this case, the elements of the regression vector \( \varphi(t) \) are known and the parameters can be estimated using the least squares method.

Instead of a linear predictor model as in (3.8) and (3.12), we can use a non-linear model. One class of non-linear models can be written as
\[
\hat{y}(t|\theta) = g(\varphi(t), \theta). \tag{3.15}
\]

It is common to write \( g(\varphi(t), \theta) \) as a function expansion
\[
g(\varphi(t), \theta) = \sum_{k=1}^{d} \alpha_k g_k(\varphi) \tag{3.16}
\]
where \( g_k \) are the basis functions and \( \alpha_k \) constants. One common way to obtain \( g_k \) is by parametrizing a mother basis function \( \kappa \) as
\[
g_k(\varphi) = \kappa(\varphi, \beta_k, \gamma_k) \tag{3.17}
\]
where \( \beta_k \) is related to scale and \( \gamma_k \) to translation. For example, it is possible to construct \( g_k \) as a ridge function
\[
g_k(\varphi) = \kappa(\beta_k^T \varphi + \gamma_k). \tag{3.18}
\]
One way to choose \( \kappa \) is by a smooth version of a step such as the sigmoid function
\[
\kappa(x) = \sigma(x) = \frac{1}{1 + e^{-x}}. \tag{3.19}
\]
See Figure 3.3 for an illustration of the sigmoid function. The combination of (3.16), (3.18) and (3.19) is called a one hidden-layer feedforward sigmoid neural network. This is a special case of the general NARX class. As in linear models, one method to fit a model to data is by minimizing the quadratic norm of the prediction error. See [20] for a detailed description of non-linear models and a comparison between neural networks and the standard models used for automatic control.
3.4 Model Validation

An important part of system identification is to test the validity of the model. The concept of cross-validation is to divide the measured data in two parts, estimation data and validation data. As their names indicate, the estimation data is used to estimate the model and the validation data is used to validate the model. One validation technique is to compare a model’s output from simulation with the validation data. A performance measure called model fit in percent is defined as

\[
M_f = \left( 1 - \frac{\|\mathbf{e}(t, \hat{\theta}_N)\|}{\|\mathbf{y}(t) - \bar{y}\|} \right) \cdot 100\%
\]  

(3.20)

where \(\bar{y}\) is the mean value of \(\mathbf{y}\) and \(\|\cdot\|\) is the \(\ell_2\)-norm. Another validation technique is by analyzing the prediction error, \(e\), as in (3.2), which should ideally be independent of the input signal. If not, the model has missed to describe some of the system dynamics. It is common to define the cross-covariance as

\[
\hat{R}_{\mathbf{e}\mathbf{u}}^N(\tau) = \frac{1}{N} \sum_{t=1}^{N} e(t)u(t - \tau).
\]

(3.21)

If \(e(t)\) and \(u(t)\) are independent, the cross-covariance should be close to zero [17].
The chapter explains the design and implementation of the different parts of the system. It starts with an overview of the system followed by the dataset used for the system identification. Different models are estimated and results from the validation are presented. The final parameter values of the proposed model are shown. The chapter follows with an explanation of the user behaviour model and ends with a discussion.

4.1 CPU Load Control System

In this thesis project, a simulation environment of the CPU load control system has been developed in MATLAB and SIMULINK. It consists of four subsystems: acceptance rate check, a user behaviour model, a CPU load model and an overload protection algorithm. The CPU load controller in the true RBS system and an alternative CPU load controller is implemented in the overload protection subsystem.

The acceptance rate check subsystem evaluates if the RBS system has the capacity to accept a connection request and uses a time window with window length $W_L$. This subsystem is event-driven, which means that the decision whether to accept or reject a connection request is instantaneous. This means that it is not possible to queue connection requests. A limit on the maximum number of requests that can be accepted per time window is defined as the acceptance rate, $a_r$. A counter, $S_{\text{acc},W_L}$, is defined as the cumulative number of connection requests that are accepted in a time window. At the beginning of each new time window, $S_{\text{acc},W_L}$ is reset. A connection request is accepted if

$$S_{\text{req},W_L} < a_r$$  \hspace{1cm} (4.1)
Figure 4.1: A new connection request, $x$, is accepted, $\exists$, if the counter $S_{\text{req}, W_L}$ has not exceeded $a_r$. The acceptance rate, $a_r$, is 3 in this example and only 3 connection requests can be accepted in each time window, $W_L$. In the first interval $[0, W_L]$ there are 6 requests and the last 3 connection requests are rejected. In the following interval, 3 out of 4 connection requests are accepted.

and rejected if

$$S_{\text{req}, W_L} \geq a_r.$$  \hspace{1cm} (4.2)

This is illustrated in Figure 4.1. The user behaviour model describes how a UE acts when trying to connect to an RBS and when transmitting data, that is, when a UE sends a connection request in the idle state and when and how much data to transmit in the connected state. It also specifies when a UE is allowed to try to reconnect in the event of a rejected connection request. The user behaviour model is described further in Section 4.3.

The CPU load model describes how the total number of accepted connection requests in each time interval, $T_s$, affects the CPU load. The CPU load is measured instantaneously and is denoted $y$ and has the sample time $T_s$. Note that in the real RBS system, the CPU load is influenced by other control signals and the overload protection algorithm itself, see Section 1.3 and Section 2.3. A saturation is used to limit the CPU load between 0 and 100%. The CPU load model is further described in Section 4.2.1.

Connection requests from UEs are not controllable and can in a control application be considered as disturbances. The overload protection uses $y$ in order to adjust $a_r$ dynamically and is described further in Section 5.1. Thus, in a control application, $a_r$ is a control signal. Another controllable signal is the window length, $W_L$, but, $W_L$ is assumed to be constant in this thesis. Note that $T_s$ not necessarily is equal to $W_L$. See Figure 4.2 for an illustration of the implemented CPU load control system.
4.2 CPU Load Model

4.2.1 Experimental Method and Data

Datasets have been provided by Ericsson AB. Simulated UEs with a preset behaviour have been used to collect CPU load data from real E-NODEBS. An RBS processes the UEs without knowing whether they are real or simulated. The datasets used in this thesis contain RRC messages and CPU load measurements and are measured in a multicore digital unit. For a digital unit with a multicore architecture, only the highest loaded core is measured. All UEs in a cell are distributed to a specific core. The UEs in the datasets are simulated with an even distribution across the cells. Thereby, all cores should be equally loaded. See Figure 4.3 for an illustration. The datasets are described in Table 4.1.

The RRC messages have a sample time of 10ns while the CPU load measurements have sample time $T_s = 100ms$. Therefore, $100ms$ is chosen as the step time in the simulation. Because of the different sample times, the datasets are preprocessed before identifying the models. Given a CPU load measurement, the total number
Figure 4.3: In this example, three directed antennas cover a cell each. A cell is often a 120° sector from the RBS, that is a geographical region. All UEs in a cell are distributed to a specific core. The UEs in the datasets are simulated with an even distribution across the cells. Thereby, all cores should be loaded equally.

of accepted connection requests is accumulated in the interval between the last CPU load measurement and the current. The input signal to the CPU load model is denoted \( n_{\text{acc},T_s} \) and the output signal is \( y \). See Figure 4.4 for an illustration.

4.2.2 Modelling Results

Choosing a model structure is for many applications a recursive process, so also in this thesis. Most linear models such as ARX, ARMAX and BJ were considered at first and different parameter values were examined. The dataset \( d_1 \) was divided into two parts, estimation data and validation data. Models were estimated with the estimation data and cross-validated with the validation data. For several of the models, the model fit was sufficient. However, all the estimated models had the problem that they could not reproduce the CPU load when validated with the dataset \( d_2 \). See Figure 4.5 for an ARX \([8 \ 8 \ 0]\) model validated with dataset \( d_2 \).

Table 4.1: Two datasets are used to estimate a model. The datasets contain information about the control signalling of each UE, that is, the time of a new connection request and when it is accepted or rejected. The datasets also contain the measured CPU load with sample time \( T_s \).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_1 )</td>
<td>Divided into two parts, estimation data and validation data.</td>
</tr>
<tr>
<td>( d_2 )</td>
<td>Completely new dataset, used to validate the model.</td>
</tr>
</tbody>
</table>
4.2 CPU Load Model

The CPU load measurements are sampled every $T_s = 100\text{ms}$. RRC messages such as accepted connection requests, $x$, are sampled at a higher rate and the datasets are therefore preprocessed. Given a CPU load measurement, $n_{\text{acc}, T_s}$ is the total number of accepted connection requests in the interval between the last CPU load measurement and the current.

![Diagram](image)

**Figure 4.4:** The CPU load measurements are sampled every $T_s = 100\text{ms}$. RRC messages such as accepted connection requests, $x$, are sampled at a higher rate and the datasets are therefore preprocessed. Given a CPU load measurement, $n_{\text{acc}, T_s}$ is the total number of accepted connection requests in the interval between the last CPU load measurement and the current.

Non-linear ARX Models

The result of the estimated linear models was unsuccessful and led to the evaluation of non-linear models and in particularly the non-linear ARX (NARX) model. A NARX $[10 \ 3 \ 5]$ model proved to be more promising and could somewhat reproduce the CPU load during simulation. See Figure 4.6 for a comparison of the measured and simulated CPU load and Table 4.2 for statistics. Different CPU load intervals are evaluated and it can be seen that the estimated NARX $[10 \ 3 \ 5]$ model has problem to reproduce the CPU load in the upper intervals but the major dynamics can to some extent be followed. The load intervals are chosen according to the implementation of the current controller. As will be described in Section 5.1, these load intervals are used by the CPU load controller to adjust the acceptance rate, $a_r$.

During simulation we noticed a problem with the NARX $[10 \ 3 \ 5]$ model. If we set $a_r = 0$ and thereby the input signal $n_{\text{acc}, T_s} = 0$ for $t > t^*$, we still get an output $\hat{y} > 0$ for $t > t^*$. This an unrealistic behaviour of a CPU and thus undesirable for our application. If there are no processes running, the CPU load should go towards zero. This flaw is illustrated in Figure 4.7. At overload, as described in Section 5.1, $a_r$ are set to zero. Because the datasets are not measured at high load, $a_r$ is never set to zero. This means that the model is extrapolated and needs to be able to describe behaviours not present in the datasets.
4. System Overview and Modelling

Figure 4.5: Comparison between the measured CPU load, $y$, and simulated CPU load, $\hat{y}$, using a ARX $[8 \ 8 \ 0]$ model with dataset $d_2$. The major dynamics are not captured by the model. The scaling of the $y$ axis is removed at the request of Ericsson AB.

Figure 4.6: Comparison between measured CPU load, $y$, and simulated CPU load, $\hat{y}$, using a NARX $[10 \ 3 \ 5]$ model with dataset $d_2$. The major dynamics are captured better by this model. However, $\hat{y}$ is out of phase. The scaling of the $y$ axis is removed at the request of Ericsson AB.
Table 4.2: Comparison of total time in each load interval between measured CPU load, \( y \), and simulated CPU load, \( \hat{y} \), using a NARX [10 3 5] model on dataset \( d_1 \) and \( d_2 \). The load interval limits are defined by \( c_1 \), \( c_2 \) and \( c_3 \) (%), where \( c_3 > c_2 > c_1 \). The simulation indicate that the dynamics of the system can to some extent be explained by the model.

<table>
<thead>
<tr>
<th>Interval</th>
<th>( d_1(%) )</th>
<th>( d_2(%) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y &gt; c_3 )</td>
<td>27.91</td>
<td>11.96</td>
</tr>
<tr>
<td>( \hat{y} &gt; c_3 )</td>
<td>18.27</td>
<td>4.65</td>
</tr>
<tr>
<td>( c_2 &lt; y \leq c_3 )</td>
<td>4.98</td>
<td>5.65</td>
</tr>
<tr>
<td>( c_2 &lt; \hat{y} \leq c_3 )</td>
<td>10.30</td>
<td>16.61</td>
</tr>
<tr>
<td>( c_1 &lt; y \leq c_2 )</td>
<td>11.63</td>
<td>16.28</td>
</tr>
<tr>
<td>( c_1 &lt; \hat{y} \leq c_2 )</td>
<td>12.62</td>
<td>22.92</td>
</tr>
<tr>
<td>( y \leq c_1 )</td>
<td>55.48</td>
<td>66.11</td>
</tr>
<tr>
<td>( \hat{y} \leq c_1 )</td>
<td>58.80</td>
<td>58.81</td>
</tr>
<tr>
<td>STD, ( \sigma_y )</td>
<td>25.92</td>
<td>21.24</td>
</tr>
<tr>
<td>STD, ( \sigma_{\hat{y}} )</td>
<td>24.67</td>
<td>19.91</td>
</tr>
</tbody>
</table>

Figure 4.7: Simulation with a NARX [10 3 5] model. The request rate, \( n_{\text{req},T_s} \), is defined as the total number of UEs trying to connect to the RBS per time \( T_s \). The total number of UEs that are rejected per time \( T_s \) is defined as \( n_{\text{req},T_s} \). At time \( t = t^* \) we set \( a_r = 0 \) so that all connection requests are rejected. Still, the CPU load, \( \hat{y} \), stabilizes on a CPU load significant larger than zero. This is unrealistic. The scaling of the \( y \) axis is removed at the request of Ericsson AB.
Output-Error Model

Motivated by the problems with the estimated NARX [10 3 5] model, we search for another model with low model order, ability to capture the major dynamics and such that $\hat{y} \to 0$ when $t \to \infty$ and $n_{\text{acc},T_s} = 0$. These requirements were met with a linear OE model that has the structure

$$y(t) = \frac{B(q)}{F(q)} u(t) + e(t).$$  \hfill (4.3)

The OE model is a special case of the BJ model that was discarded with the motivation that it could not reproduce the CPU load during validation with the second dataset, $d_2$. The difference between the two is that the BJ model also uses old values of the noise to predict the output. A saturation is used to limit the CPU load between 0 and 100%. See Figure 4.8 for a comparison of the measured and simulated CPU load and Table 4.3 for statistics. With the OE [10 10 0] model, $\hat{y} \to 0$ when $a_r = 0$, as can be seen in Figure 4.9. This is a more realistic behaviour than the one in Figure 4.7.

**Proposed CPU Load Model**

The proposed CPU load model is the estimated OE [10 10 0] model (4.3). The estimated polynomials $B$ and $F$ are given by

$$B(q) = 0.6328 + 0.6747q^{-1} + 0.6391q^{-2} + 0.3308q^{-3} + 0.1458q^{-4} + 0.2377q^{-5} - 0.02721q^{-6} + 0.2027q^{-7} + 0.2832q^{-8} + 0.5096q^{-9}$$  \hfill (4.4)

and

$$F(q) = 1 + 0.3636q^{-1} - 0.09428q^{-2} - 0.08429q^{-3} + 0.3722q^{-4} + 0.1077q^{-5} - 0.2245q^{-6} + 0.2989q^{-7} + 0.2746q^{-8} - 0.2529q^{-9} - 0.8303q^{-10}.$$  \hfill (4.5)

The proposed model is motivated by the fact that it could reproduce the CPU load during validation with the second dataset, $d_2$ according to Table 4.3 as well as having no static error for $a_r = 0$ as in Figure 4.9. As can be seen in Figure 4.10 there is no risk of pole-zero cancellations for the relatively high model order of OE [10 10 0]. This indicates that over-modelling is not an issue.

**4.3 User Behaviour Model**

The user behaviour model is a stochastic model of when and how large amounts of data a UE would like to transmit. The data bursts do not influence the CPU load themselves, but the transmission pattern of a UE is the underlying cause of when a new connection request is sent. Thereby, if the connection request is accepted, the CPU load is influenced. The model has been estimated by Ericsson AB by analyzing transmission patterns of UEs in the mobile telecommunication network.
A UE in the idle state cannot transmit data. When a UE is in the connected state, it can continue to send new data bursts (a number of bytes of data). If the time since the last data burst is larger than a threshold, the UE will be released from the RBS. The threshold is called the inactivity time (IAT) and has the purpose to save battery time of the UE and to clear memory in the RBS. After the release, a UE has to initiate a new connection request in order to send additional data. See Figure 4.11 for an illustration of the communication flow for a specific UE from a simulation.

A data burst can be divided into smaller pieces, called chunks. This occurs when a UE requests to transmit a larger amount of data than is possible in one time interval due to the transmission speed. The transmission speed depends on the radio quality, which in turn depends on for instance the number of UEs connected to the RBS and if there are obstacles in the radio path. In this thesis, the transmission speed is assumed to be constant since it is the major traffic dynamics that is of interest. See Figure 4.12 for an illustration of the concept of chunks.

In order to model when a UE requests to transmit data we introduce the concept of burst inter-arrival time (BIAT). BIAT is the time between the end of a burst (that is, when all chunks are transmitted) and the start of a new burst and it is modeled as a stochastic variable with a distribution $p(t_{BIAT})$, obtained from Ericsson AB’s traffic analysis. At the end of a burst, a new BIAT realization is chosen and if the new BIAT is larger than the threshold, IAT, the UE will be released. See Figure
Table 4.3: Comparison of total time in each load interval between measured CPU load, $y$, and simulated CPU load, $\hat{y}$, using a saturated OE $[10 10 0]$ model on dataset $d_1$ and $d_2$. The load interval limits are defined by $c_1$, $c_2$ and $c_3$ (%) where $c_3 > c_2 > c_1$. The OE $[10 10 0]$ model can, compared to a NARX $[10 3 5]$ model, reproduce the dynamics of the system better.

<table>
<thead>
<tr>
<th>Interval</th>
<th>$d_1$ (%)</th>
<th>$d_2$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y &gt; c_3$</td>
<td>27.91</td>
<td>11.96</td>
</tr>
<tr>
<td>$\hat{y} &gt; c_3$</td>
<td>24.58</td>
<td>14.29</td>
</tr>
<tr>
<td>$c_2 &lt; y \leq c_3$</td>
<td>4.98</td>
<td>5.65</td>
</tr>
<tr>
<td>$c_2 &lt; \hat{y} \leq c_3$</td>
<td>6.98</td>
<td>10.63</td>
</tr>
<tr>
<td>$c_1 &lt; y \leq c_2$</td>
<td>11.63</td>
<td>16.28</td>
</tr>
<tr>
<td>$c_1 &lt; \hat{y} \leq c_2$</td>
<td>11.96</td>
<td>16.61</td>
</tr>
<tr>
<td>$y \leq c_1$</td>
<td>55.48</td>
<td>66.11</td>
</tr>
<tr>
<td>$\hat{y} \leq c_1$</td>
<td>56.48</td>
<td>58.47</td>
</tr>
<tr>
<td>STD, $\sigma_y$</td>
<td>25.92</td>
<td>21.24</td>
</tr>
<tr>
<td>STD, $\sigma_{\hat{y}}$</td>
<td>29.14</td>
<td>24.34</td>
</tr>
</tbody>
</table>

Figure 4.9: Simulation with a saturated OE $[10 10 0]$ model. At time $t = t^*$ we set $a_r = 0$ so that all connection requests are rejected. The CPU load, $\hat{y}$ goes to zero. The scaling of the $y$ axis is removed at the request of Ericsson AB.
Figure 4.10: The proposed OE [10 10 0] model has a relatively high order. However, as can be seen from the figure, the confidence regions do not overlap. Thus, there seems to be no risk of pole-zero cancellations.

4.13 for the BIAT distribution. The burst size, $B_s$, that is the amount of bytes a UE want to transmit in one burst, is also modeled as a stochastic variable with a distribution $p(B_s)$, obtained from Ericsson AB’s traffic analysis.

With the probabilistic models $p(t_{BIAT})$ and $p(B_s)$ it is possible to describe the communication flow of a UE. Each UE will choose a new BIAT and $B_s$ independently of each other. Thus, the UEs will have unique data transmission patterns. This is necessary in order to have a realistic number of connection requests over time in the simulations. The only tunable parameter is the total number of UEs in the simulation. The stochastic behaviour of the user behaviour model is illustrated in Figure 4.15.
4.4 Discussion

Different dynamic models have been evaluated and it has been shown that a saturated OE model is suitable to reproduce the measured CPU load from the datasets. The saturation is used to limit the CPU load between 0 and 100 (%). The measured load is from the highest loaded core. This means that a request processed on another core will not affect the CPU load measurement but will be part of the input that is used to identify the model. For an even distribution of UEs over the cells the load on each core should in average be equal. Thus, the model should be able to reproduce the CPU load of the highest loaded core for an even distribution of UEs in the cells.

A statistical user behaviour model has been implemented in a simulation environment. The model is produced by Ericsson AB and is used to simulate transmission patterns of UEs. The resulting number of connected users is in line with measurements from Ericsson AB, and it is thus possible to simulate a population of UEs using this model.

Figure 4.11: The communication flow of a UE during simulation. The UE is initially in the idle state. At time $t_1$ the UE is accepted into the system and transitions to the connected state. Then the UE sends a couple of data bursts. At time $t_2$ the UE is released from the RBS due to inactivity and transitions to the idle state. The UE is accepted once again at time $t_3$ in order to send new bursts of data and transitions to the connected state.
4.4 Discussion

Given the data transmission speed for a UE, there is a maximum limit on the amount of data that is possible to transmit in one time interval. If the maximum limit is reached, the data burst will be divided into chunks. In this example, a UE requests to send 20kB of data at time $t_1$. The data burst is divided into three chunks, with 8kB, 8kB and 4kB data each and transmitted at time $t_1$, $t_2$ and $t_3$, respectively.

A better dataset would have had CPU load measurements with the same sample time as the RRC messages together with information about which core a specific message is processed on. That would have made it possible to estimate a multiple input multiple output model or one model per core. The dataset used to estimate the model includes only a few rejects which are not used as input. All RRC messages affect the CPU load and it would have been interesting with a dataset with a higher average CPU load and thereby a larger number of rejected connection requests. We are interested in a model that reproduces the behaviour of the CPU load control system at high load. Thereby it is not as important that the model is consistent at low load. With a high-load dataset, the simulations would have reflected the real RBS load control system better. Measurements from Ericsson AB show that the connection requests that are accepted affect the CPU load more than the other RRC messages. Thus, the dynamics of the RBS load control system can still be modelled.
Figure 4.13: A new BIAT is chosen according to a distribution $p(t_{BIAT})$. Note that the x axis is truncated.

Figure 4.14: A new burst size is chosen according to a distribution $p(B_s)$. Note that the x axis is truncated.
**Figure 4.15:** Ten different UEs transmit data according to the BIAT distribution. As can be seen the UEs have unique data transmission patterns. This is due to the stochastic nature of the user behaviour model.
The following chapter explains the control design for the CPU load control system. It starts with the current version of the CPU load controller followed by an alternative controller. The controllers are evaluated on two different test cases. A fitness function is used to measure the performance.

5.1 Current CPU Load Controller

The purpose of the CPU load controller is to dynamically adjust the acceptance rate, \( a_r \), in order to avoid overload in the CPU. The current CPU load controller consists of three subsystems: an FIR filter, a CPU state machine and an actuator that updates the acceptance rate, \( a_r \). See Figure 5.1 for an illustration of the current CPU load controller.

The FIR filter uses the 10 last CPU load measurements and calculates a moving average, \( \bar{y} \), as

\[
\bar{y}(k) = \frac{1}{10} \sum_{i=0}^{9} y(k - i).
\]

The CPU load is measured every \( T_s \) seconds and the FIR filter calculates a new \( \bar{y} \) in every sampling instant. Depending on \( y \) and \( \bar{y} \), the CPU state is updated in every \( T_s \) sampling instant. The CPU states correspond to the load interval of \( \bar{y} \) and are defined in Table 5.1. The overload state is a safety mechanism to avoid the worst case scenario of a full system reset. For this reason, it is important that the CPU state is updated as often as possible. In the overload state, \( a_r \) is set to zero, that is, all new connection requests are rejected. Thus, it is important to adjust \( a_r \) such
Control Design

**Figure 5.1:** The current overload protection consists of three subsystems in order to dynamically change the acceptance rate, $a_r$.

**Table 5.1:** The CPU states for the current controller. The state is changed depending on the current state, the latest CPU load measurement and the average CPU load. The load interval limits are defined by $c_1$, $c_2$ and $c_3(\%)$ where $c_3 > c_2 > c_1$.

<table>
<thead>
<tr>
<th>CPU state</th>
<th>Load interval (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overload</td>
<td>$\bar{y} &gt; c_3$</td>
</tr>
<tr>
<td>Very high</td>
<td>$c_2 &lt; \bar{y} \leq c_3$</td>
</tr>
<tr>
<td>High</td>
<td>$c_1 &lt; \bar{y} \leq c_2$</td>
</tr>
<tr>
<td>Not high</td>
<td>$\bar{y} \leq c_1$</td>
</tr>
</tbody>
</table>

that the overload state is avoided.

The actuator updates the acceptance rate, $a_r$, every $8T_s$ based on the CPU state. If the CPU load state is overload or very high, $a_r$ is decreased, and reversed, if the CPU load is in the high or not high state $a_r$ is increased. The increase and decrease is constant and denoted $\delta$. The variable $a_r$ is limited to the interval $[a_{r,\text{min}}, a_{r,\text{max}}]$. Even though no new connection requests are accepted in the overload state, the acceptance rate $a_r$ is still updated. By doing so, $a_r$ will be lower once the CPU load state transitions to the very high state. The high and not high CPU load states are both handled the same in the CPU load control system and could for this thesis be merged. However, in the real version, these states are used to extract statistics. As one can see, the current version of the CPU load control system updates $a_r$ less frequently than $\bar{y}$ and the CPU state. One reason for doing this, could be to control $a_r$ more smoothly. As will be described in Section 5.2 an alternative controller is proposed that uses another approach. This alternative controller adjusts $a_r$ more frequently but with a smaller step than $\delta$. See Figure 5.2 for an illustration of the current actuator.

Controlling the CPU load by updating $a_r$ based on the CPU load state can be viewed as an integration. The current version of the CPU load controller allows a smaller and smaller control signal every $8T_s$ if the CPU load is too high. And
5.2 Alternative PI CPU Load Controller

$\ar := \ar - \delta$

$\ar := \ar + \delta$

**Figure 5.2:** The current actuator either increases or decreases the $\ar$ by $\delta$.

the reversed, if the CPU load is lower than the capacity of the CPU, $\ar$ is increased every $8T_s$. In order to see that the current version of the CPU load controller can be viewed as an integration more clearly we will rewrite it as in Algorithm 1. We also define a special case of the sign operator as

$$\tilde{\text{sgn}}(t) = \begin{cases} 1, & t \geq 0 \\ -1, & t < 0 \end{cases} \quad (5.2)$$

**Algorithm 1** Current CPU load controller

Run every $8T_s$ seconds:

$\bar{y}_k = \frac{1}{10} \sum_{i=0}^{9} y(k-i)$

$\bar{\ar}_{r,k} = \bar{\ar}_{r,k-1} + \delta \tilde{\text{sgn}}(c_2 - \bar{y}_k)$

if $\bar{y}_k > c_3$ then

$\ar_{r,k} = 0$

else

$\ar_{r,k} = \bar{\ar}_{r,k}$

end if

5.2 Alternative PI CPU Load Controller

An alternative CPU load controller is implemented to adjust $\ar$ every sampling interval. This is eight times faster than the current CPU load controller and is motivated by that the CPU load is measured every $T_s$ second. By doing so, we adjust $\ar$ as soon as we have new information of the CPU load. The current CPU load controller can as mentioned be seen as an integration. The increase and
Figure 5.3: An alternative PI CPU load controller is implemented to adjust $a_r$. The CPU load is filtered through an FIR filter to get the moving average CPU load, $\bar{y}$. A reference, $r$, is chosen to $c_2$ as in Table 5.1. Both the integrating part, $I_k$ and the total of the proportional part and the integrating part, $P_k + I_k$, are saturated in order to limit how much it is possible to adjust $a_r$.

decrease of $a_r$ is constant so it is natural to try an alternative controller that makes $a_r$ proportional to the integrated error, $e(t) = r(t) - \bar{y}(t)$. The reference, $r$, is chosen to $c_2$ as in Table 5.1. As the current CPU load controller, the moving average CPU load, $\bar{y}$, is used in order to not control $a_r$ too aggressively. However, we also add a proportional part to be able to adjust $a_r$ faster. Both the integrating part, $I_k$ and the total of the proportional part and the integrating part, $P_k + I_k$ are saturated in order to limit how much it is possible to adjust $a_r$. The safety mechanism for the overload state is remaining. The alternative PI CPU load controller is shown in Algorithm 2 and in Figure 5.3.

**Algorithm 2 Alternative PI CPU load controller**

<table>
<thead>
<tr>
<th>Run every $T_s$ second:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bar{y}<em>k = \frac{1}{10} \sum</em>{i=0}^{9} y(k - i)$</td>
</tr>
<tr>
<td>$e_k = c_2 - \bar{y}_k$</td>
</tr>
<tr>
<td>$P_k = K_p e_k$</td>
</tr>
<tr>
<td>$b = a_{r,\text{min}} - a_{r,\text{max}}$</td>
</tr>
<tr>
<td>$I_k = \min(\max(I_{k-1} + K_i e_k, b), 0)$</td>
</tr>
<tr>
<td>$\tilde{a}<em>{r,k} = a</em>{r,\text{max}} + \min(\max(P_k + I_k, b), 0)$</td>
</tr>
<tr>
<td><strong>if</strong> $\bar{y}_k &gt; c_3$ <strong>then</strong></td>
</tr>
<tr>
<td>$a_{r,k} = 0$</td>
</tr>
<tr>
<td><strong>else</strong></td>
</tr>
<tr>
<td>$a_{r,k} = \tilde{a}_{r,k}$</td>
</tr>
<tr>
<td><strong>end if</strong></td>
</tr>
</tbody>
</table>

By setting the proportional constant, $K_p$, to zero and and only adjust the integrating constant, $K_i$, it should be possible to get similar results as the current CPU load controller, since this is an I controller. The alternative CPU load controller adjusts $a_r$ eight times faster than the current CPU load controller so $K_i$ should be about eight times smaller than $\delta$. 
5.3 Evaluation

5.3.1 Test Cases

Two test cases are defined to evaluate the controllers. In the first test case the reference of the controller changes repeatedly in order to force the controller to adjust $a_r$ more often. In the second test case an external load is added to the CPU. This happens when other, non-UE-related tasks are executed on the CPU, for instance operations and maintenance in the RBS.

The test case simulations are initialized as follows: Two preparatory simulations...
are done with 18000 and 25000 UEs, respectively. These simulations are done with the current controller, and our only interest in them is to sample the states of each individual UE when the system is in steady state. These samples are then used for initialization of our two test cases. See Figure 5.5 for an illustration of the simulation with 18000 UEs where each state of the UEs is saved at steady state at time \( t^* \). The total number of UEs in a simulation affects how many connection requests there are per sampling interval. This in turn affects how many connection requests that are accepted per sampling interval and thereby influences the CPU load. By simulating both 18000 and 25000 UEs we evaluate if the CPU load controller can handle different intensities of accepted connection request. Both 18000 and 25000 UEs are in this context a lot and used to evaluate the CPU load control system at high load.

**Test Case 1**

In the first test case, the CPU load reference of the controllers is changed repeatedly for a period of time with a square wave. For the current controller, this is handled by lowering the load threshold in the very high and high state respectively. The P and PI controller are more straightforward and are handled by lowering the reference value. For both controllers, the reference, \( c_2 \), is changed. The length of the first test case is 4000 s and the reference is changed from \( t = 1000 \) s. See Figure 5.6 for an illustration of the change of reference.

**Test Case 2**

In the second test case, we add independent external load at different times. The load varies between 6 and 15 (\%). The same realization of external load is used for all tests. The length of the second test case is 4000 s and the external load is added from \( t = 1000 \) s. See Figure 5.7 for an illustration of the external load.

**5.3.2 Fitness Function**

In order to evaluate the different test cases and parameter settings, a fitness function, \( F_f \), is defined. This function consists of two parts: The first part, \( F_{L,\text{rej}} \), weights the number of rejected connection requests. The second part, \( F_{Q,\text{over}} \), weights the CPU load overshoot and is defined as the CPU load that is larger than the reference load. As mentioned earlier the number of rejected connection requests should be minimized in order to increase the quality of service. The CPU load overshoot should be minimized as well in order to not block resources intended for other activities. One could define a fitness function in several ways. In this thesis, the fitness function weights the rejected connection requests linearly but the overshoot quadratically. Rejected connection requests are an unwanted cost no matter when they occur. For example, two rejected connection requests at \( t_1 \) is as bad as if the first connection request was rejected at time \( t_1 \) and the second at time \( t_2 \). The total of two end users will be rejected no matter. For this reason, the fitness function weights the rejected connection requests linearly. The CPU load overshoot is also an unwanted cost. Large deviations from the reference are
worse because then the CPU load gets closer to the overload state. If it is possible to decrease the oscillation, one could increase the reference and by that increase the number of UEs in the RBS. Therefore, the fitness function weights the CPU load overshoot quadratically. The fitness function, $F_f$ is evaluated between time 1000s and 4000s and is defined as

$$F_f = \frac{F_{L,\text{rej}} + F_{Q,\text{over}}}{2} \quad (5.3)$$

where

$$F_{L,\text{rej}} = Ts \frac{1}{N} \sum_{k=0}^{N} n_{\text{rej},T_s}(k) \quad (5.4)$$

and

$$F_{Q,\text{over}} = Ts \frac{1}{N} \sum_{k=0}^{N} (\max(0, \hat{y}(k) - r(k)))^2 \quad (5.5)$$

A $F_f$ close to zero is preferable since this would mean that there are no rejected connection requests or CPU load overshoots. For simplicity, $F_f$ is normalized with respect to the performance of the current controller.

### 5.3.3 Simulation Results

#### Test Case 1

Different parameters of the PI controller have been tested in simulations and the results are summarized in Table 5.2 and Table 5.3. In Table 5.2 an I controller is tuned to get similar results as with the current controller. As can be seen, an I controller can be tuned to get similar performance as the current controller. Both parts of $F_f$, the rejected connection requests and the CPU load overshoots are close to 1. Results from simulation of Test Case 1 are illustrated in Figure 5.8. The performance is increased with the alternative PI CPU load controller. For $K_p = 1.31$ and $K_i = 0.05$, the performance of $F_{L,\text{rej}}$ and $F_{Q,\text{over}}$ is increased by 14% and 64%, respectively. The corresponding increase of performance in Table 5.2 is 13% and 82%, respectively.

#### Test Case 2

All the different parameters of the PI controller that have been tested in Test Case 1 have also been tested in Test Case 2. The results from Test Case 2 are summarized in Table 5.4 and Table 5.5.

As can be seen in Table 5.4, the performance is increased with the alternative PI CPU load controller. For $K_p = 1.31$ and $K_i = 0.05$, the performance of $F_{L,\text{rej}}$ and $F_{Q,\text{over}}$ is increased by 16% and 12%, respectively. The corresponding increase of performance in Table 5.5 is 9% and 26%, respectively.
Table 5.2: Results from simulation of Test Case 1 with 18000 UES. As can be seen it is possible to get similar results as the current controller with an i controller. The highest increase in performance for each part is marked in grey. The values chosen based on the overall result are marked in bold.

<table>
<thead>
<tr>
<th>Controller</th>
<th>$K_p$</th>
<th>$K_i$</th>
<th>$F_f$</th>
<th>$F_{L,rej}$</th>
<th>$F_{Q,over}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>-</td>
<td>-</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>I</td>
<td>-</td>
<td>0.08</td>
<td>0.78</td>
<td>0.92</td>
<td>0.63</td>
</tr>
<tr>
<td>I</td>
<td>-</td>
<td>0.1055</td>
<td>0.94</td>
<td>0.91</td>
<td>0.97</td>
</tr>
<tr>
<td>I</td>
<td>-</td>
<td>0.106</td>
<td>0.99</td>
<td>0.94</td>
<td>1.03</td>
</tr>
<tr>
<td>I</td>
<td>-</td>
<td>0.107</td>
<td>1</td>
<td>0.93</td>
<td>1.06</td>
</tr>
<tr>
<td>I</td>
<td>-</td>
<td>0.108</td>
<td>0.97</td>
<td>0.93</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>-</td>
<td>0.125</td>
<td>1.94</td>
<td>0.92</td>
<td>2.96</td>
</tr>
<tr>
<td>I</td>
<td>-</td>
<td>0.14</td>
<td>3.73</td>
<td>0.97</td>
<td>6.48</td>
</tr>
<tr>
<td>PI</td>
<td>1</td>
<td>0.05</td>
<td>0.61</td>
<td>0.84</td>
<td>0.38</td>
</tr>
<tr>
<td>PI</td>
<td>1.31</td>
<td>0.05</td>
<td>0.61</td>
<td>0.86</td>
<td>0.36</td>
</tr>
<tr>
<td>PI</td>
<td>1.31</td>
<td>0.08</td>
<td>0.56</td>
<td>0.86</td>
<td>0.26</td>
</tr>
<tr>
<td>PI</td>
<td>1.31</td>
<td>0.1</td>
<td>0.58</td>
<td>0.88</td>
<td>0.27</td>
</tr>
<tr>
<td>PI</td>
<td>1.6</td>
<td>0.05</td>
<td>0.6</td>
<td>0.86</td>
<td>0.34</td>
</tr>
<tr>
<td>PI</td>
<td>1.6</td>
<td>0.08</td>
<td>0.56</td>
<td>0.86</td>
<td>0.26</td>
</tr>
</tbody>
</table>

Table 5.3: Results from simulation of Test Case 1 with 25000 UES. The highest increase in performance for each part is marked in grey. The values chosen based on the overall result are marked in bold.

<table>
<thead>
<tr>
<th>Controller</th>
<th>$K_p$</th>
<th>$K_i$</th>
<th>$F_f$</th>
<th>$F_{L,rej}$</th>
<th>$F_{Q,over}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>-</td>
<td>-</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>PI</td>
<td>1</td>
<td>0.05</td>
<td>0.54</td>
<td>0.88</td>
<td>0.2</td>
</tr>
<tr>
<td>PI</td>
<td>1.31</td>
<td>0.05</td>
<td>0.53</td>
<td>0.87</td>
<td>0.18</td>
</tr>
<tr>
<td>PI</td>
<td>1.31</td>
<td>0.08</td>
<td>0.53</td>
<td>0.88</td>
<td>0.18</td>
</tr>
<tr>
<td>PI</td>
<td>1.31</td>
<td>0.1</td>
<td>0.54</td>
<td>0.89</td>
<td>0.19</td>
</tr>
<tr>
<td>PI</td>
<td>1.6</td>
<td>0.05</td>
<td>0.53</td>
<td>0.88</td>
<td>0.17</td>
</tr>
<tr>
<td>PI</td>
<td>1.6</td>
<td>0.08</td>
<td>0.53</td>
<td>0.89</td>
<td>0.17</td>
</tr>
</tbody>
</table>

Table 5.4: Results from simulation of Test Case 2 with 18000 UES. The highest increase in performance for each part is marked in grey. The values chosen based on the overall result are marked in bold.

<table>
<thead>
<tr>
<th>Controller</th>
<th>$K_p$</th>
<th>$K_i$</th>
<th>$F_f$</th>
<th>$F_{L,rej}$</th>
<th>$F_{Q,over}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>-</td>
<td>-</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>PI</td>
<td>1</td>
<td>0.05</td>
<td>0.79</td>
<td>0.76</td>
<td>0.83</td>
</tr>
<tr>
<td>PI</td>
<td>1.31</td>
<td>0.05</td>
<td>0.86</td>
<td>0.84</td>
<td>0.88</td>
</tr>
<tr>
<td>PI</td>
<td>1.31</td>
<td>0.08</td>
<td>0.9</td>
<td>0.8</td>
<td>1</td>
</tr>
<tr>
<td>PI</td>
<td>1.31</td>
<td>0.1</td>
<td>0.92</td>
<td>1</td>
<td>0.84</td>
</tr>
<tr>
<td>PI</td>
<td>1.6</td>
<td>0.05</td>
<td>0.82</td>
<td>0.75</td>
<td>0.89</td>
</tr>
<tr>
<td>PI</td>
<td>1.6</td>
<td>0.08</td>
<td>0.88</td>
<td>0.94</td>
<td>0.81</td>
</tr>
</tbody>
</table>
5.4 Discussion

The current version of the CPU load controller and an alternative PI CPU load controller have been implemented in MATLAB. An I controller has been tuned to get similar results as the current controller. The results indicates that it is possible to see the current CPU load controller as an integrating controller. Thereby, it should be possible to rewrite the current version of the CPU load controller in order to make it more straightforward to tune it with methods from automatic control.

An alternative PI CPU load controller is proposed and evaluated. Given the proposed fitness function, the performance is improved, both in terms of lower amount of rejected connection requests and decreased CPU load overshoot. With the parameter values $K_p = 1.31$ and $K_i = 0.05$, the performance is improved in two test cases for two configurations of UEs. In the worst simulation case the number of rejected connection requests is decreased by 9% and the CPU load overshoot by 12%. That is, an increase in performance is still made. In order to confirm that the alternative CPU load controller is more suitable for CPU load control, a wider range of test cases need to be simulated.
Figure 5.5: Simulation with 18000 UEs. The average CPU load (top left), the number of UEs in the connected state (top right) and the number of requests per $T_s = 100ms$ (bottom) are in steady state at time $t^*$. The internal states for each UE at $t = t^*$ are saved and used to initialise the UEs in the test case simulations.
5.4 Discussion

Figure 5.6: In the first test case, the CPU load reference is changed repeatedly for a period of time according to a square wave.

Figure 5.7: In the second test case, an external disturbance is added. Note that the x axis is truncated and that only the interval [700, 1400] is shown.
Figure 5.8: Result from simulation of Test Case 1 with 18000 UEs. A PI controller with $K_p = 1.31$ and $K_i = 0.05$ is used.

Table 5.5: Results from simulation of Test Case 2 with 25000 UEs. The highest increase in performance for each part is marked in grey. The values chosen based on the overall result are marked in bold.

<table>
<thead>
<tr>
<th>Controller</th>
<th>$K_p$</th>
<th>$K_i$</th>
<th>$F_f$</th>
<th>$F_{L,\text{rej}}$</th>
<th>$F_{Q,\text{over}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>-</td>
<td>-</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>PI</td>
<td>1</td>
<td>0.05</td>
<td>0.82</td>
<td>0.93</td>
<td>0.7</td>
</tr>
<tr>
<td>PI</td>
<td>1.31</td>
<td>0.05</td>
<td>0.83</td>
<td>0.91</td>
<td>0.74</td>
</tr>
<tr>
<td>PI</td>
<td>1.31</td>
<td>0.08</td>
<td>0.86</td>
<td>0.93</td>
<td>0.78</td>
</tr>
<tr>
<td>PI</td>
<td>1.31</td>
<td>0.1</td>
<td>0.9</td>
<td>0.94</td>
<td>0.85</td>
</tr>
<tr>
<td>PI</td>
<td>1.6</td>
<td>0.05</td>
<td>0.85</td>
<td>0.94</td>
<td>0.76</td>
</tr>
<tr>
<td>PI</td>
<td>1.6</td>
<td>0.08</td>
<td>0.86</td>
<td>0.93</td>
<td>0.79</td>
</tr>
</tbody>
</table>
6

Conclusions

The RBS CPU load control system is complex and an approximated version has been implemented in this thesis. By analyzing more control signals such as rejected connection requests it would be possible to estimate a model that would reflect the real system better. However, even though approximations have been made, this thesis indicates that it is possible to simulate the major dynamics of the CPU load control system with the proposed model. This can be used by Ericsson AB as a pilot study to improve the CPU load control system. The current CPU load controller has previously not been developed with theories from automatic control and this thesis has given the tools to view the problem from another perspective. By improving the CPU load controller, the mobile telecommunication network would be more accessible for the end users, which would mean economic benefits and also a higher connectivity in the networked society.

It is not straightforward to propose a fitness function to measure the performance. Throughout this thesis, we have been talking about rejected connection requests as if they were all alike. However, an emergency call undoubtedly has higher priority than an everyday conversation. The fitness function is essentially a business decision that has to be in line with what is commercially viable. For instance, would society accept that a mobile device is rejected because it is old and energy inefficient in order to promote environmentally friendly mobile devices?
Bibliography


7.

ericsson.com/ourportfolio/telecom-operators/

[4] Karl Johan Åström and Tore Hägglund. Advanced PID Control. ISA-

Mokhtari. CPU load prediction model for distributed computing. In Eighth
International Symposium on Parallel & Distributed Computing, page 39,

Mokhtari. CPU load prediction using neuro-fuzzy and Bayesian inferences.
Neurocomputing, volume 74:1606 – 1616, 2011. ISSN 0925-2312. Cited on
page 3.

on pages 3, 6, and 7.

on pages 3, 6, and 7.


Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-komersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnt som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/

© Joachim Larsson