An Erlang Implementation of a Scalable Node B Control Unit

Author: Adrià Casas Escoda
Supervisor and Examiner: Johan Montelius, KTH.
Industrial Supervisor: Anders Danne, Ericsson AB.

Master’s Thesis
School of Information and Communication Technology
KTH Royal Institute of Technology
Abstract

The demand of mobile data traffic is increasing due to the popularization of advanced mobile devices such as smartphones and tablets and to the generalization of the use of mobile Internet. The Node B is one of the main elements of the control plane of the UMTS network. It is responsible for the tasks directly connected to the radio interface and provides the physical radio link between the mobile devices and the network.

This master thesis presents a design of the Node B control unit that can handle multiple requests concurrently and scale both by the number of cores and the cards. Additionally, analyzes the suitability of using a high level language such as Erlang for implementing the Node B control unit. To achieve these objectives, a prototype of the Node B control unit that can handle requests concurrently and scale by the number of cores and cards has been designed and implemented with Erlang.

The developed prototype shows that implementing a concurrent and scalable Node B control unit with Erlang is completely feasible and the tests that have been carried out demonstrate that the performance and scalability of the system are good.

Furthermore, some realistic deployment scenarios of an Erlang implementation of the Node B control unit over the real hardware used in the Radio Base Station at Ericsson have been discussed and they show that it is completely possible to use Erlang for implementing the Node B control unit.
## Contents

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>3</td>
</tr>
<tr>
<td>List of Tables</td>
<td>5</td>
</tr>
</tbody>
</table>

### 1 Introduction
- 1.1 Thesis Motivation                      | 6    |
- 1.2 Problem Description                    | 6    |
- 1.3 Method                                 | 7    |
- 1.4 Delimitations                          | 7    |
- 1.5 Outline of the Thesis                  | 7    |

### 2 Theoretical Background
- 2.1 UMTS                                    | 9    |
- 2.2 SCTP                                    | 15   |
- 2.3 Erlang                                  | 16   |

### 3 Analysis and Design
- 3.1 Introduction                            | 17   |
- 3.2 Functional Requirements                 | 17   |
- 3.3 Non-functional Requirements             | 18   |
- 3.4 Programming Language                    | 18   |
- 3.5 Architecture                            | 18   |
- 3.6 Design Strategies for Developing a Scalable System | 21   |
- 3.7 Use Cases                               | 24   |

### 4 Implementation
- 4.1 Communication Between Units            | 29   |
- 4.2 Implementation Description             | 29   |
- 4.3 Some Implementation Issues             | 34   |

### 5 Performance Analysis
- 5.1 Testing Tools                          | 38   |
- 5.2 Testing Environment                     | 39   |
- 5.3 System Performance with Different Number of Cores | 39   |
- 5.4 System Performance with Different Number of Cards | 43   |
- 5.5 Analysis of the Performance of the Used Protocols | 45   |
6 Future Work
   6.1 Addition of New Functionalities ............................................. 51
   6.2 Possible Deployment Scenarios ............................................... 52

7 Conclusions ................................................................. 54

Bibliography ............................................................... 56
List of Figures

2.1 UMTS architecture .......................................................... 10
2.2 UTRAN Architecture ....................................................... 10
2.3 UTRAN Channels ............................................................ 13
2.4 Example of NBAP procedure ............................................. 14

3.1 System Architecture ....................................................... 19
3.2 Radio Link Setup pipeline ................................................. 22
3.3 Radio Link Setup parallel pipeline ...................................... 22
3.4 Distribution of the elements of the system when using multiple cards 24
3.5 System Initialization sequence diagram .................................. 25
3.6 Audit phase sequence diagram .......................................... 26
3.7 Radio Link Setup sequence diagram 1 .................................. 27
3.8 Radio Link Setup sequence diagram 2 .................................. 27
3.9 Radio Link Deletion sequence diagram 1 .............................. 28
3.10 Radio Link Deletion sequence diagram 2 .............................. 28

4.1 System architecture diagram ............................................. 30
4.2 Time needed to handle 10,000 operations with different request handling assign-
ment strategies ................................................................. 35
4.3 Time needed to handle 10,000 operations with different number of cards distribut-
ing or not the NBAP Handlers ................................................ 36

5.1 Testing elements distribution ............................................. 39
5.2 Median time needed to complete a test consisting of 10,000 operations with dif-
ferent number of cores. Recall that for each Radio Link Setup request, 10 ms of sleep time and 1 ms of busy work time are performed .............................................. 40
5.3 Median time needed to complete the processing of each kind of procedure in the
Node B and the Radio Network Controller .................................. 41
5.4 Median time needed to handle a Radio Link Setup request in the Node B .......... 42
5.5 Distribution of the time needed to handle a Radio Link Setup request in the Node
B ................................................................................................ 43
5.6 Median time needed to complete a test consisting of 10,000 operations with dif-
ferent number of cards. Recall that for each Radio Link Setup request, 10 ms of sleep time and 1 ms of busy work time are performed .............................................. 44
5.7 Median time needed to encode/decode 40,000 Radio Link Setup request messages
with different number of cores .................................................. 47
5.8 Comparison of LINX performance from Erlang and C .......................... 48
LIST OF FIGURES

5.9 Time needed to send 10,000 messages with Erlang and LINX between the same
and different machines. ................................................................. 49
5.10 Time needed to send 10,000 messages with Erlang and LINX between the same
machine. ..................................................................................... 50
6.1 Architecture of the deployment using a Control card. ............................... 52
6.2 Architecture of the deployment using a single card with OSE and Linux. .... 52
6.3 Architecture of the deployment using a single card with Linux. ................. 53
List of Tables

5.1 Median time needed to complete a test consisting of 10,000 operations and achieved speed-up with different number of cores. ........................................ 40
5.2 Time needed to handle a Radio Link Setup request in the Node B in milliseconds. 42
5.3 Median time needed to complete a test consisting of 10,000 operations and achieved speed-up with different number of cards. ................................. 44
5.4 Median time needed to send 5,000 SCTP messages between two machines. .... 46
5.5 Median time needed to send 10,000 messages of different sizes between two ma-
chines using LINX. ................................................................. 48
Chapter 1

Introduction

1.1 Thesis Motivation

The demand of mobile data traffic is increasing dramatically due to the popularization of advanced mobile devices such as smartphones and tablets and to the generalization of the use of mobile Internet. It is anticipated that the average annual growth rate of mobile data traffic will remain in the order of 50 - 100% for the next decade. According to the UMTS Forum the total worldwide mobile traffic will be of more than 127 EB in year 2020, representing 33 times more traffic compared with year 2010. Furthermore, the “always on” property of new devices means that the network has to be able to handle more attached users at any given time, which also puts demands on control plane capacity.

This high expected increase in data traffic is a challenge for communication systems providers and mobile operators as they have to continuously improve the capacity of the mobile network in order to cope with this incredible growth of mobile data traffic [1, 2, 3].

In this thesis is presented and analyzed a concurrent and scalable implementation of the Node B, one of the main elements of the control plane of the UMTS network. The Node B is responsible for the tasks directly connected to the radio interface and provides the physical radio link between the mobile devices and the network.

1.2 Problem Description

The increasing number of radio connections required by new mobile services makes necessary to increase the mobile network control plane capacity. The aim of this master thesis is to present a design of the Node B control unit that can handle multiple requests concurrently and scale both by the number of cores and the cards. Additionally, it is also the purpose of this thesis to determine the suitability of using a high level language such as Erlang for implementing the Node B control unit.

The following questions will be considered during the realization of this master thesis:

- How can the Node B control unit be designed and implemented in order to handle requests concurrently and scale with increasing number of cores and cards?

- Which is the expected performance of an Erlang implementation of the Node B control unit when executing the system on multi-core and distributed architectures and which are

\footnote{With card it is referred the hardware system where the Node B is executed}
CHAPTER 1. INTRODUCTION

the bottlenecks of the implementation?

- Which are the benefits and drawbacks of using a high level language such as Erlang to implement the Node B control unit?

To achieve these objectives and give an answer to these questions a prototype of the Node B control unit that handles requests concurrently and scales by the number of cores and cards will be designed and implemented with Erlang and its performance will be analyzed.

1.3 Method

This master thesis requires a deep understanding of the role of the Node B within the UMTS network and of the NBAP protocol. Also a certain level of knowledge of the whole UMTS network and the architecture of the UTRAN is required. For this reason, the literature regarding these topics will be studied.

In order to design a prototype of the Node B control unit the current implementation of the Node B at Ericsson will be studied and analyzed. Afterwards, the analysis of the requirements of the prototype will be carried out and the design of the system will be performed. Finally the designed prototype will be implemented with Erlang.

The developed prototype will be tested and the obtained results will be analyzed in order to determine the performance and scalability of the system. A general analysis of the expected performance of an Erlang implementation of a scalable Node B control unit will be done using the obtained results.

1.4 Delimitations

A real Node B is a big and complex system that performs a wide range of operations. For this reason, implementing a complete prototype of the Node B is out of the scope of this project. The developed system will be a prototype of the Node B control unit that will be able to perform a small subset of all the functionalities done by a real Node B. This implemented prototype should be sufficiently complete to perform performance studies of its performance and allow the extraction of valid conclusions. This means that sufficiently many types of NBAP messages should be handled so that all aspects of the system can be evaluated.

1.5 Outline of the Thesis

First of all, the theoretical background about UMTS and Erlang needed to fully understand this master thesis is presented in Chapter 2. In that chapter are explained the main parts of the UMTS network and a detailed description of the UTRAN architecture and the NBAP protocol is given. It also contains a brief description of Erlang and its main features. Chapter 3 describes the requirements and the design of the developed prototype and explains the main design decisions that have been taken and its motivation. In Chapter 4 is provided a detailed description of the implementation of the system and the solutions taken to solve some of the presented problems are explained. The performance of the implemented prototype is evaluated in chapter 5, where figures with the performance of the system with different configurations are shown and their results analyzed and compared. In chapter 6 is discussed the possible evolution of the work done in this thesis and how the implemented prototype can be expanded and deployed on the
real Node B hardware used at Ericsson. Finally, the conclusions and a summary of the most important parts of the project are presented in chapter 7.
Chapter 2

Theoretical Background

The purpose of this chapter is to give to the reader necessary knowledge to fully understand the problem that is discussed in this thesis and the proposed solution. An overview of the UMTS network is given, focusing on the aspects more relevant for this project like the architecture of the UTRAN and the NBAP protocol. Also the Erlang programming language and its main features are presented.

2.1 UMTS

2.1.1 Introduction

The Universal Mobile Telecommunications System (UMTS) is one of the third generation mobile technologies that provides high bandwidth data and voice services to mobile users. The development of standards for 3G cellular networks started in 1995 although it was not until year 1998 that 3rd Generation Partnership Project (3GPP) was created to develop a global standard for third generation mobile networks.

The first version of the UMTS specification, called Release 99, provided significant performance improvements over previous generation mobile networks, including data transfer rates of up to 2 Mb/s and Quality of Service (QoS) capabilities. It was published on 1999 [4].

2.1.2 Architecture

UMTS can be divided into three logical blocks, each one with different responsibilities.

The first block is the UMTS network terminal, that is called User Equipment (UE). The UE has two different parts, the UMTS Service Identity Module (USIM) and the Mobile Device. The USIM contains user-specific information and the authentication key that permits user access to the network. In the Mobile Device is located the protocol stack of the radio interface as well as the elements of the user interface.

The second functional block is the UTMS Terrestrial Radio Access Network (UTRAN), the fixed network infrastructure that contains the facilities for transmitting over radio. The UTRAN is divided into Radio Network Subsystems. A Radio Network Subsystem consists of a set of radio elements and their corresponding controlling element.

Finally the Core Network is the network responsible for transporting user data to its destination. The Core Network has a multitude of switching systems as well as gateways to other
networks, such as the fixed phone network or the Internet. It also includes databases used for mobility management, user management and billing.

Between each of these elements are defined standard interfaces. Between the UE and the UTRAN the interface is called Uu. The other major interface is Iu, located between the UTRAN and the Core Network [7, 6].

2.1.3 UMTS Terrestrial Radio Access Network

The UTRAN is divided into individual Radio Network Subsystems (RBS) connected to the Core Network through the Iu interface. A Radio Network Subsystem is composed by one Radio Network Controller (RNC) and one or more Node Bs, each one of them managing a set of cells. The Node B is connected to the Radio Network Controller through the Iub interface.
The UTRAN is responsible for the control and handling of the radio resources and allows data and signaling traffic exchange between the UE and the Core Network. Its main task is to create and maintain Radio Bearers. It also controls some functions related to UE mobility and network access.

A Radio Bearer is a connection between the UE and the Radio Network Controller carrying user data. With Radio Bearers, the different elements of the Core Network are given a way of communicating with the UE, releasing them of the responsibility of taking care of radio communication aspects. A Radio Bearer has to fulfill the Quality of Service (QoS) requirements specified by the Core Network. The UTRAN has to create and maintain Radio Bearers so that the QoS requirements are always fulfilled. Some of the specific tasks of the UTRAN are:

- Admission Control: The purpose of the admission control is to admit or deny new users or new radio links. The admission control should avoid overload situations and it bases its decisions on interference and resource measurements.
- Congestion Control: The task of the congestion control is to detect and handle situations when the system is reaching overload or is in an overload situation. This means that some part of the network has run out - or will soon run out - of resources.
- Handover: This function manages the mobility of the UE between different Cells.
- Ciphering: Encrypts exchanged information and is located between Radio Network Controller and UE.
- Radio resource configuration and operation: This function performs the configuration of the radio network resources, for example Cells and common transport channels, and takes resources into and out of operation [5, 6].

2.1.3.1 Radio Network Controller

The Radio Network Controller is the switching and controlling element of the UTRAN and is located between the Iub and In interface. A Radio Network Controller controls all the tasks related to data transmission over the radio interface. These tasks are grouped under the concept of Radio Resource Management. The Radio Network Controller has essentially the following responsibilities:

- Call admission control: There is a large number of channels at the radio interface, although not all of them can be used at the same time due to interference problems. The Radio Network Controller decides if the interference level that will be achieved after a requested channel has been occupied is acceptable. Otherwise it rejects the request.
- Radio resource management: Manages the radio resources of all the attached Cells.
- Radio bearer management: Sets up and disconnects Radio Bearers and manages QoS.
- Power Control: It is essential for the operation of the radio network that the transmission power of all the users is controlled. The actual fast control process takes places in the Node B but the target control values are established by the Radio Network Controller.
- Handover: Based on the measurement values given by the Node B and the UE, the Radio Network Controller detects when a different Cell would be better for a current connection. When the Radio Network Controller decides to perform a handover, it is responsible for the signaling with the new Cell and for informing the UE about the new channel.
As explained above, the Iub is the interface between the Radio Network Controller and the Node B. This interface is divided into two planes:

- **Control plane**: Used for transmission of radio network control messages. The protocol used is NBAP, explained in section 2.1.5.
- **User Plane**: Used for carry user and control data that is not part of the NBAP protocol [5].

### 2.1.3.2 Node B

The Node B handles the tasks directly connected to the radio interface and provides the physical radio link between the UE and the network. It organizes transmission and reception of data across the radio interface and codifies data for transmitting it over the radio. The tasks of the Node B are directed by its Controlling Radio Network Controller. The Node B is responsible for:

- **Radio resource management**: Manages the radio resources of its Cells.
- **Power Control**: Measures the interference levels in the radio interface, compares it with the target value set by the Radio Network Controller and, if necessary, triggers changes in the transmission power of a UE.
- **Measurement report**: The Radio Network Controller has to know the current situation in each Cell so it can take the best decisions in handover, power control and call admission. For this reason, the Node B carries out periodically measurements of the connection quality and interference levels and communicate the results to the Radio Network Controller [7, 5].

### 2.1.3.3 Node B Structure

The internal structure of the Node B is vendor-dependent, but its logical structure is standardized. A Node B is characterized by the following elements:

- The logical resources provided to the UTRAN, grouped into Cells.
- The dedicated channels that have been established on the Node B.
- The common transport channels that the Node B provides to the Radio Network Controller.

From the point of view of the Radio Network Controller these resources are grouped into two elements, common transport and a number of Traffic Termination Points. The common transport represents the transport channels that are common for all the UEs in the cell and those used for initial access to the network.

A Traffic Termination Point consists of a number of Node B Communication Contexts. A Node B Communication Context contains all the radio resources associated with a UE in that Node B once the UE has set a dedicated connection. Thus, one Communication Context contains a set of dedicated uplink and downlink channels.

From the point of view of the Radio Network Controller the Node B consists of a set of logical entities called Cells. A Cell is the smallest division of the radio network and is identified by its identification number (Cell Id.) [8, 7].
2.1.4 Kinds of channels

Different kinds of channels exist within the UMTS network. Taking into account between which entities is set a channel, the following kinds of channels are defined:

- **Physical Channel**: A radio link between the Node B and the UE.
- **Transport bearer**: Link between the Radio Network Controller and the Node B.
- **Transport Channel**: Channel between the Radio Network Controller and the UE. A radio bearer is carried over a transport channel and a physical channel.
- **Logical channel**: Channel between the Core Network and the UE. A logical channel is carried over a radio bearer.

![UTRAN Channels](image)

Figure 2.3: UTRAN Channels

Bearing in mind who uses a certain channel, the following classification can be done:

- **Common channel**: The same channel is used by all the UEs in a Cell.
- **Dedicated channel**: Each UE has its own channel.

2.1.5 NBAP

The Node B Application Part (NBAP) protocol is the communication protocol used between the Radio Network Controller and the Node B and is specified using ASN.1\(^1\). This protocol is used by the Radio Network Controller to configure and manage the Node B and setup channels between the Node B and the UE - called radio links - and between the Node B and the Radio Network Controller - called transport bearers.

A wide range of functionalities are performed using this protocol. Among this functionalities, some of the most important are:

- **Cell configuration**: Configuring the Cells of the Node B.
- **Common transport channel management**: Configuration of common transport channels in the Node B.
- **Resource event management**: Informing the Radio Network Controller about the status of the Node B resources.

\(^1\)Abstract Syntax Notation 1 (ASN.1) is a standard notation used to describe messages and specify data structures to be exchanged between different communicating entities at a high level of abstraction. ASN.1 is used by a wide range of applications such as network management, secure email and mobile telephony. Associated with ASN.1 are sets of standardized encoding rules that describe how the messages have to be encoded as they are in transit between different entities. Neither ASN.1 nor its encoding rules are tied to any computer architecture, operating system or programming language [16].
• Measurements on common and dedicated resources: Initiate measurements in the Node B and report its results.

• Radio Link Management and Supervision: Manage radio links using dedicated resources in the Node B and reporting failures and restorations.

• Reporting of general error situations.

The NBAP protocol consists of elementary procedures. An elementary procedure is a unit of interaction between the Radio Network Controller and the Node B that consists of an initiating message and possibly a response message. There are two kinds of NBAP procedures:

• Class 1: Elementary procedures with response. In this kind of procedures, the response can be of two types: successful or unsuccessful.

• Class 2: Elementary procedures without response.

In Figure 2.4 can be seen an example of a Class 1 procedure with both successful and unsuccessful responses [5, 9].

2.1.6 UTRAN procedures examples

To achieve a better understanding of the operation of UTRAN, a detailed description of the realization of some procedures performed in UTRAN is given.

2.1.6.1 Node B setup

A Node B setup is performed if a Node B is added to the network, changes its configuration or after a system restart. To make the network aware of these changes, the Node B initiates a setup procedure. The steps followed during the setup of the Node B are the following ones:
1. The Node B requests to be audited by the Radio Network Controller. This is done by sending an Audit Required NBAP message to the Radio Network Controller. The Radio Network Controller answers to this message by initiating an Audit procedure sending an Audit request message to the Node B. In response the Node B informs the Radio Network Controller about the number of Cells that manages and which local identifiers they have.

2. For each Cell of the Node B a Cell Setup NBAP procedure is performed by the Radio Network Controller. During the Cell Setup the physical common radio channels of the Cell are parametrized. If these channels are not active it is impossible for the UE to access the network via the radio interface.

3. The common transport channels are set up in each Cell of the Node B by the Radio Network Controller using the Common Transport Channel Setup NBAP procedure. For each of the common physical radio channels a transport bearer is set in the Iub interface between the Radio Network Controller and the Node B \([5, 9]\).

2.1.6.2 Radio Link Setup

This procedure is used to establish the necessary air interface resources for a dedicated channel that is related to a Communication Context in the Node B. The steps followed during the Radio Link Setup process are:

1. The Radio Network Controller initiates a Radio Link Setup NBAP procedure by sending a Radio Link Setup request message to the Node B. This message contains the Radio Network Controller Communication Context identifier, that is used by the Radio Network Controller to identify the Context that will be created, a Cell identifier, that identifies the Cell where the dedicated channels have to be set, and a Radio Link identifier. It also contains the information to configure the dedicated channels in the Node B.

2. The Node B sets the requested dedicated channels on the radio interface. Also the necessary transport bearers to carry the user plane data between the Node B and the Radio Network Controller are established.

3. The Node B sends the Radio Link Setup Successful response message to the Radio Network Controller \([5, 9]\).

2.2 SCTP

The Stream Control Transmission Protocol (SCTP) is a transport layer protocol, existing at an equivalent level with UDP and TCP. Like TCP, SCTP provides a reliable transport service, ensuring that data is transported across the network without errors and in sequence. SCTP is session oriented, meaning that a relationship between the endpoints of an SCTP association is created prior to data is being transmitted.

Unlike TCP, SCTP provides a number of functions that are critical for telephony signaling transport. The basic features of SCTP are:

- Provides reliable data transmission, detecting when data is discarded, reordered, duplicated or corrupted, and retransmitting it when necessary.

- Message oriented, meaning that in SCTP a sender sends a message in one operation, and that exact message is delivered by the receiving SCTP entity to the receiving application in a single operation.
• Multi-streaming. This means that SCTP can provide multiple streams between connection endpoints, each one with its own reliable sequenced delivered messages.
• Multi-homing support, which allows a single SCTP endpoint to have more than one IP address [11, 10].

2.3 Erlang

Erlang is a declarative language for programming concurrent and distributed systems. It was developed at Ericsson to implement large soft real-time control systems. The first version of Erlang was released in 1986. It is a dynamically typed, single assignment language that uses pattern matching for variable binding and function calling selection and has support for concurrent and distributed processes. Distributed Erlang programs can run transparently on cross-platform multi-vendor systems. The language has primitives for detecting run-time errors and for dynamic code replacement. Most of the system is written in Erlang with just the core run-time system and some low-level system calls implemented in C.

The main strength of Erlang is support for concurrency and distribution. It has a small set of primitives used to create and manage processes and communicate among them. Erlang processes are neither operating system processes nor threads, but lightweight processes and like operating system processes they have no shared memory between them. Inter-process communication works via a shared-nothing asynchronous message passing system: every process has a “mailbox”, a queue where are stored the messages that have been sent by other processes and not yet read. A message can contain any Erlang structure, including primitive types (integers, floats, characters, atoms), tuples, lists, and functions.

The distribution of processes among different nodes is almost transparent. An Erlang node is one instance of the run-time environment, usually implemented as a single process (with many threads) on Unix systems.

In 1998 Ericsson released Erlang as open source under the Erlang Public License [14, 15].
Chapter 3
Analysis and Design

3.1 Introduction
In this master thesis a prototype of the Node B control unit is designed and developed. There already exists a Node B implementation at Ericsson, and to develop this prototype the current system has been studied. The goal of this project is not to make a copy of the existing system, but to develop with Erlang a prototype of the Node B control unit that can perform a subset of the functionalities of the existing system while being able to handle requests concurrently and scale with the number of cores and cards.

In this chapter are analyzed the requirements of the system and the proposed design of the Node B control unit is presented.

3.2 Functional Requirements
The Node B has to maintain a state and perform a set of functionalities. This state consists of the following elements:

- The set of UTRAN Cells managed by the Node B, with their configuration and common channels.
- The Communication Contexts, where all the information regarding the UEs that have a radio link with the Node B is stored. This information includes the dedicated channels.
- The transport channels that the Node B has set with the radio network controller.

The Node B performs a wide range of functionalities that are requested by the Radio Network Controller using NBAP procedures. These functionalities are related to radio resource management, power control and measurement report. The developed prototype is not going to perform all the actions of a real Node B but only a subset of them related to radio resource management. These actions are the initialization of the Node B, ordered by the Radio Network Controller using the Audit and Cell Setup NBAP procedures, and the handling of the Radio Link Setup and Radio Link Deletion NBAP procedures.

It has been decided to implement the Radio Link Setup and Radio Link Deletion NBAP procedures because these are the main procedures related to radio resource management. Also because in this way the prototype is able to handle one common NBAP procedure - Radio Link Setup - and one dedicated NBAP procedure - Radio Link Deletion. This makes the design of the
prototype more extensible, as the way dedicated and common procedures are handled has some common characteristics

### 3.3 Non-functional Requirements

The main goal of this thesis is to develop an scalable prototype of the Node B control unit. The term scalability has a broad and sometimes ambiguous meaning. In the context of this thesis, scalability is defined as the ability of a system of increasing its performance when additional hardware resources are added.

In the case of this prototype, when talking about performance, we are referring to the throughput of the system, this is the number of operations handled per unit of time, not the latency of each operation.

Depending on the kind of resources that are added to the system, two kinds of scalability can be distinguished: vertical and horizontal scalability. Scale vertically (also known as scale up) is the ability of a system of improving its performance when additional hardware resources are added to the node where it is executed. Scale horizontally (or scale out) means that a system can increase its performance by distributing its execution among several nodes [17, 18].

In the concrete case of this Node B control unit implementation, the system has to be able to increase its throughput in relation to the number of cores and cards. For this reason, the most important non-functional requirement is that the system has to be able to scale horizontally and vertically.

Apart from this, another important requirement is that the system has to be extensible, meaning that it has to be possible to add new functionalities to it - for example, the handling of more types of NBAP procedures - without making significant changes on the already implemented part.

### 3.4 Programming Language

The programming language to use should not be the first aspect to consider when designing a software system, but in this case the fact of using Erlang determines the design process and makes much simpler fulfilling the main requirement of this project: scalability.

Erlang has some features that make it completely different of the most common programming languages used for implementing this kind of systems such as C, C++ or Java. The lightweight processes, that are easy and cheap to create and manage, and the integrated message passing system encourage writing in a style that minimizes shared state. Furthermore, some things that are often needed in scalable systems are straightforward in Erlang, such as implementing state machines (gen_fsm behavior), packing and unpacking protocol messages (great binary syntax) or remote failure detection (process monitors). These features make easier designing and implementing concurrent and scalable programs with Erlang than with other programming languages.

### 3.5 Architecture

The Node B prototype has been designed in order to have a system that fulfills the requirements presented above and using as a model the existing Node B implementation at Ericsson. The Node B has been divided in two separate units: the control and the operational unit.
The control unit contains all the control logic of the system and asks to the elements of the operational unit to perform the required low level operations. These low level operations are related to establishing transport channels with the Radio Network Controller or configuring the radio devices.

The aim of this thesis is to study the performance of a scalable Node B control unit, but in order to have a complete and testable system, a dummy operational unit is also designed and developed.

In Figure 3.1 is displayed an FMC block diagram\(^1\) containing the main elements of the system and their relations. Now follows a brief description of the different elements that will conform the Node B prototype.

\subsection{3.5.1 Control Unit}

There can be distinguished two set of tasks performed by the control unit, and for this reason this unit has been divided into two subsystems:

The Node Control Subsystem, which is responsible for the following tasks:

\begin{itemize}
  \item Initializing the system.
  \item Setting up and maintaining the control plane connection with the Radio Network Controller.
\end{itemize}

\(^1\)Fundamental Modeling Concepts (FMC) provides a framework to describe software systems. It strongly emphasizes the communication about software-intensive systems by using a semi-formal graphical notation that can easily be understood. Block Diagrams are used to represent the compositional structures of systems. Their main graphical elements are agents, storages and channels [19].
• Performing the ASN.1 encoding and decoding of NBAP messages.
• Managing the audit phase.

And the Channel Control Subsystem, that performs all the actions related to the handling of common and dedicated channels.

3.5.2 Node Control Subsystem

Three modules have been designed to carry out the tasks of this subsystem. These modules are the Node Controller, the Connection Manager and the NBAP Handler.

The Node Controller is responsible for initializing all the elements of the system and forwarding the NBAP requests received by the Connection Manager to a NBAP Handler. It also manages the audit phase, as this stage affects the configuration of many elements of the Node B.

The Connection Manager is responsible for the control plane connection with the Radio Network Controller. It sets and releases the connection with the Radio Network Controller and supervises the established link. Once the connection has been set, the main tasks performed by the Connection Manager are sending and receiving NBAP messages between the Radio Network Controller and the Node B.

In the UMTS specification are defined two options for implementing the control plane communication between the Node B and the Radio Network Controller: ATM\(^2\) and SCTP. As it does not exist an ATM implementation over Ethernet, it would be necessary to have special hardware for setting up an ATM connection between two computers. On the other hand there is no problem in setting up an SCTP connection between two computers as SCTP runs over IP and there exists an Erlang module to manage SCTP connections. For this reason it has been decided to implement the connection between the Radio Network Controller and the Node B using SCTP.

The third module of the Node Control Subsystem is the NBAP Handler, which performs the ASN.1 encoding and decoding of NBAP messages. It is also responsible for routing the decoded NBAP requests to the suitable module of the Channel Control Subsystem.

3.5.3 Channel Control Subsystem

This unit performs the tasks related to the management of dedicated and common channels. The elements that conform this unit are the following ones:

Cell

Is the logical representation of a UTRAN Cell in the Node B. It handles the set up of a Cell during the audit phase and stores the configuration of its common channels. Each Cell has a global identifier assigned by the Radio Network Controller during the audit phase.

Communication Context and Context Resource Handler

The Communication Context is the logical representation of a UE and is responsible for managing and storing the configuration of all the active Radio Link Sets of a UE in the Node. Each active Communication Context has an internal identifier assigned by the Node B and a global identifier assigned by the Radio Network Controller during its initialization. In the Node B there is a fixed

\(^2\)Asynchronous Transfer Mode (ATM) is a switching technique for telecommunication networks. It uses asynchronous time-division multiplexing, and it encodes data into small, fixed-sized packets. ATM provides data link services that run over OSI physical layer.
set of Communication Contexts that is created when the Node B is started. Each time that a Communication Context is needed - that happens when a Radio Link Setup is being performed - the Context Resource Handler is used to get a free one. The Context Resource Handler is responsible for creating and managing the set Communication Contexts of the Node B.

**Radio Link Set**

A Radio Link Set is a set of Radio Links of a UE in the Node B that share certain characteristics - have a common generation of transmit power control commands in the downlink. In the prototype the Radio Link Set module is responsible for establishing and releasing Radio Links and storing its configuration.

**Radio Resource Handler**

The Radio Resource Handler manages the radio devices of the Base Band and provides the necessary radio resources - Base Base devices - to the Radio Link Set when a new Radio Link has to be established.

**Transport Resource Handler**

The Transport Resource Handler is responsible for setting up and releasing transport bearers for user plane communication between the Radio Network Controller and the Node B.

### 3.5.4 Operational Unit

The operational unit has two elements, the Base Band Process and the CPP. The Base Band Process controls the radio resources of the Node B and is responsible for initializing, setting up and releasing the Base Band devices. The CPP manages the user plane connections between the Node B and the Radio Network Controller that are established using transport bearers.

### 3.6 Design Strategies for Developing a Scalable System

As explained in section 3.3, one of the main requirements of the system is that it has to be able to scale horizontally and vertically. This is to increase its performance with relation to the number of cores and cards\(^3\). There have been followed different strategies to achieve both kinds of scalability. Here are presented the design strategies followed to achieve vertical and horizontal scalability.

#### 3.6.1 Scale with the Number of Cores

To make a system scale with the number of available cores it is necessary that many independent processes can be simultaneously running, so the computational power of all the cores is used.

The main task of the Node B is to handle NBAP requests. If the Node B handles requests sequentially it is impossible to achieve a high degree of concurrency as most of the steps needed to complete the handling of a request can not be done in parallel due to data dependencies. The solution for achieving a high level of concurrency is handling multiple requests simultaneously.

\(^3\)When using the term card in the context of the developed system we are referring to a logical card, represented by one or more Erlang nodes, not to a physical one. An Erlang node is one instance of the Erlang run-time environment. Each Erlang node can be located in a different physical machine.
In order to handle multiple requests concurrently, the Node B should not wait to finish the handling of one request to start handling the next one. To achieve this the process of handling a request has been divided in a series of sequential steps, each one carried out by a different element, following a pipelined architecture.

In Figure 3.2 is showed the process pipeline that has to follow a Radio Link Setup request since it is received in the Node B until the response is sent to the Radio Network Controller together with a brief description of the actions performed in each step of the pipeline.

![Figure 3.2: Radio Link Setup pipeline.](image)

First of all the request is received by the Connection Manager, that forwards it to the NBAP Handler, where the request is decoded. The decoded request is sent to the Cell where the Radio Link has to be set. Then a free Communication Context receives the NBAP message and performs the necessary actions to set the requested Radio Link and builds the response message. Finally the Communication Context encodes the response and the Connection Manager sends it to the Radio Network Controller.

The fact that a single entity, the Connection Manager, appears two times in the pipeline is not a problem because the amount of time needed to do its job is much smaller than the time spent in any other steps of the pipeline.

With this design there can be simultaneously handled as many requests as the number of steps necessary to complete its processing. To achieve an even higher degree of concurrency, when possible, the elements of the pipeline can be replicated in order to handle in each of the steps of the pipeline multiple requests.

Erlang provides a very suitable way of implementing a system with this pattern. Each entity is an Erlang module, and each Erlang module can have many instances, each of them being a different Erlang process. The communication between the different processes is done using the message passing interface provided by Erlang.

![Figure 3.3: Radio Link Setup parallel pipeline.](image)

In Figure 3.3 is shown the process pipeline with the elements replicated. As can be observed in the diagram, not all the elements of the pipeline are replicated. This is the case of the Connection Manager, that is not replicated because the Node B can only set one control plane connection with the Radio Network Controller.
In the other hand, some of the elements that are part of the pipeline are already replicated. In the Node B there are a set of Cells - one for each UTRAN Cell managed by the Node - and a set of Communication Contexts - at least one for each UE that has established a Radio Link with the Node B.

One of the most time consuming parts of handling a request is the decoding of the received message - done by a NBAP Handler - and the encoding of the response - done by a Communication Context. Furthermore, the decoding and encoding of a NBAP message is completely independent of the other tasks being performed by the Node B. For both these reasons, the encoding and decoding of multiple messages can be done in parallel.

To decode multiple requests in parallel, the NBAP Handler is replicated in order to have multiple active NBAP Handlers in the Node. As there are already multiple active Communication Contexts, it is not necessary to introduce any further modification in order to encode multiple responses concurrently.

### 3.6.2 Scale with the Number of Cards

It is a requirement of the system to be horizontally scalable, this means that the prototype has to be able to improve its performance when more computational nodes are available.

It is not necessary to introduce any major change to the proposed design to achieve this objective. The chosen solution consists of replicating almost all the elements of the Node B in each of the cards. As every card has its own radio resources, each one will be responsible for managing a set of UTRAN Cells.

The main restriction to make the Node B scale by the number of cards is that although the system is deployed among multiple cards it still has to be seen as a single logical Node B for the UTRAN. Among other things, this implies that only one control plane connection can be set with the Radio Network Controller.

In the proposed design, two elements are not replicated. These elements are the Node Controller and the Connection Manager. The first one is not replicated because it is responsible for initializing and managing all the elements of the system, regardless if they are distributed among different cards or not. The Connection Manager can not be replicated because, as said before, only one control plane connection can be set with the Radio Network Controller, so only one Connection Manager is needed.

It has been considered if it would be better to replicate the NBAP Handler or not. Replicating it has the positive effect of distributing the decoding of the requests among the different cards, but on the other hand, it increases the amount of data that is necessary to send between the cards. The reason for this is that all the NBAP messages are received in the card that has the Connection Manager, and to decide which card has to handle a request it is necessary to previously decode it. So with this option, in the worst case, a request will be received in the card that has the Connection Manager, sent to another card to perform the decoding of the message and finally forwarded to the card which is the actual target of the request.

A third option has also been considered. This option consists of doing a pre-decoding of all the requests in the main card in order to obtain the information needed to determine the target card, and then do the complete decoding of the message there. In this way, the requests will always be directly send to the target card. The problem of this solution is that there are many kinds of NBAP messages, each of them with different structures and it would be very complicated and time consuming to implement a method to do this pre-decoding in a fast way. For this reason this possibility has been rejected.

Between the other two options, it is difficult to determine which is the better one as its performance may depend on the number of cards and the capacity of the connection between
them, information that is not known at the moment. It has been decided to distribute the NBAP Handlers among the different cards, in order to achieve a better load distribution among the cards.

![Diagram of system elements distribution](image)

Figure 3.4: Distribution of the elements of the system when using multiple cards.

One of the most interesting features of Erlang is that communication between processes running on local and remote nodes is done exactly in the same way. This means that the elements of the Node B that interact with processes of different nodes such as the Node Controller or the NBAP Handlers do not need to be aware of the physical location of the different elements of the system. This fact makes simple implementing a system with this structure.

### 3.7 Use Cases

A description of the most important use cases performed by the system is presented. The explained use cases are the initialization of the system, the audit phase and the handling of radio link setup and radio link deletion NBAP procedures.

In order to provide a graphical representation of the use cases, several sequence diagrams are presented. Each box of the sequence diagram is an Erlang process, and the arrows between them represent Erlang messages.

#### 3.7.1 System Initialization

The first module created when is the Node Controller that is responsible for starting the rest of the elements of the system. Then the Connection Manager is initialized, which sets the connection with the Radio Network Controller. The Operational unit, the Channel Control Subsystem and a set of NBAP Handlers are set up in each card once the connection has been established.

First of all, the Operational unit is started. This unit contains the Base Band Process and the CPP. Then all the elements of the Channel Control Subsystem are created. The Radio Resource Handler is the first process started. It asks to the Base Band Process of its card to inform about the characteristics of the available Base Band Devices just after its initialization. Then the Transport Resource Handler is started and afterwards the Context Resource Handler.

The Context Resource Handler creates the set of Communication Contexts and returns a list containing the PIDs of the Contexts. The NBAP Handlers use this list to route the incoming
Radio Link Deletion requests to a Context. Then a Cell process is created for each UTRAN Cell managed by the card. Finally, the set of NBAP Handlers of the card are started.

Once all the cards have been initialized, is built a ring formed by the Context Resource Handler of each card. This ring enables the Context Resource Handlers of the different cards to communicate between them.

Finally, the Node B asks the Radio Network Controller to start the audit phase by sending an Audit Required NBAP message when all the elements of the system have been initialized.

3.7.2 Audit Phase

This use case is initialized when an Audit Request NBAP message is received in the Connection Handler. The Connection Handler forwards the message to the Node Controller, which manages the Audit Phase. The Node Controller creates the response message that contains information about the UTRAN Cells managed by the Node B. The response is encoded and sent to the Radio Network Controller by the Connection Handler.

Then a Cell Setup Request NBAP message is received for each Cell of the Node B. The message is received by the Connection Handler and forwarded to the Node Controller. The Cell Setup Request message contains the Global Cell Identifier that has to be assigned to a Cell. The request is sent to the Cell process that manages the Cell that is being configured. Finally, the Node Controller creates and encodes the response message and sends it to the Radio Network Controller.

The set of Cells with its global identifier and the set of Communication Contexts are delivered to the NBAP Handlers once all the Cells have been configured. This is not done until this point due to the fact that the NBAP Handlers need to know the global identifier of each Cell in order to route the Radio Link Setup requests to the correct one.
3.7.3 Radio Link Setup

This use case starts when a Radio Link Setup request message is received in the Connection Handler. The message is forwarded to the Node Controller that assigns the handling of the request to one of the available NBAP Handlers. The Handler decodes the message and forwards it to the Cell with Global Identifier indicated in the NBAP message.

When the Cell receives the message, it requests a free Communication Context to the Context Resource Handler and creates a new Radio Link Set giving it access to the resources of the Cell. The created Radio Link Set together with the message is delivered to the provided Context, which is responsible for setting the Radio Link and building the Radio Link Setup response message.
When the Context receives the request it asks the provided Radio Link Set to establish the requested Radio Link. The Radio Link Set uses the Radio Resource Handler to allocate the necessary radio resources for the Radio Link. Then it asks the Base Band Process to set up the devices allocated by the Radio Resource Handler. Finally, the Transport Resource Handler allocates the necessary Transport Bearers that are set using the CPP.

Once this has been done the Communication Context builds the response with the information provided by the Radio Link Set. Finally, the response message is encoded and forwarded to the Connection Handler, which sends it to the Radio Network Controller.

### 3.7.4 Radio Link Deletion

A Radio Link Deletion is performed when a Radio Link Deletion NBAP message is received from the Radio Network Controller. The initial steps of this use case are the same of the Radio Link Setup use case.

The Connection Handler receives the message and forwards it to the Node Controller, that selects a NBAP Handler to decode the message. Once the message has been decoded it is forwarded to the Communicative Context indicated in the NBAP message.
The Communication Context sends the deletion request to all its Radio Link Sets. Each Radio Link Set is responsible for deleting the Radio Links indicated in the request that are owned by itself. To delete a Radio Link it asks to the Transport Resource Handler to free the used Transport Bearers and Base Band Devices. Then it informs to the Radio Resource Handler of the freed Devices. If the Radio Link Set is empty after deleting the requested Radio Links, it is also deleted.

When all the requested Radio Links have been deleted, the Context builds the Radio Link Deletion response and forwards it to the Connection Handler that sends the response to the Radio Network Controller.

Finally, if all the Radio Link Sets of the Communication Context have been deleted, the Context informs to the Context Resource Handler that is free.
Chapter 4

Implementation

In this chapter is presented and analyzed the implementation of the prototype and the most relevant issues related to the implementation of the system are discussed.

4.1 Communication Between Units

The Node B is divided into two units - the Control and the Operational unit - and it is necessary to have a way of communicating the elements of both units. This communication could be easily done using Erlang messages, but this solution has an important problem. It will make very difficult to substitute the dummy Operational unit used in the prototype by the real one existing in the current Node B implementation, as the real Operational unit is not implemented with Erlang.

The Node B real system at Ericsson runs under Enea OSE, a real-time Operating System developed by Enea. The Inter Process Communication (IPC) in OSE is done using LINX, an open technology developed by Enea for distributed inter process communication. LINX is platform and interconnection independent and scales well to large systems with any topology [12, 13].

Taking profit of the fact that LINX is platform independent and there is support for LINX on Linux, it has been decided to communicate the elements of both units using this technology. This decision offers two important advantages:

- The developed system is more similar to the real system as it uses the same inter process communication technology.
- It would be easier to integrate the prototype with the real Node B hardware.

4.2 Implementation Description

In Figure 4.1 is shown a diagram with all the elements that conform the implemented prototype. Each box in the diagram represents an Erlang process. Two new modules have been added in relation to the design step in order to manage LINX communication, the Base Band Router and the LINX Manager.

The implemented prototype consists of around 2,000 manually written lines of Erlang code. Additionally, in order to generate the Erlang code responsible to perform the ASN.1 decoding and encoding of the NBAP messages, the ASN.1 specification of the NBAP protocol has been used. This specification consists of 47,000 lines of ASN.1 code and has been extracted from [9].
Now follows a brief description of the implementation of each of the modules of the system.

4.2.1 Node Control Subsystem

Node Controller

This module is responsible for starting the whole system, managing the audit phase and assigning the handling of the received NBAP messages to a suitable NBAP Handler. It is implemented using the Erlang behavior Finite State Machine (gen_fsm). The reason for choosing this behavior is that it simplifies handling the different types of messages that can be received in the Node B depending on its state.

During the system initialization the Operational unit and the Channel Control Subsystem are created for each card. The number of cards is a configuration parameter and it does not have any relation with the available hardware. Each card is represented by two Erlang nodes, one containing the Operational unit and the other one containing the Channel Control Subsystem.

Starting each element in a different Erlang node gives a high degree of freedom when deciding how to distribute the system among different machines, as each Erlang node can be started in a different machine and the communication between the processes of each node is always done in the same way, independently of its physical location.

Once the system has been initialized and the configuration of all the Cells managed by the Node B has been performed the prototype changes to the state “open” and remains in this state, waiting for receiving NBAP requests, until the system is stopped or an error occurs.
There are two NBAP procedures that can be handled by the implemented prototype in this state, Radio Link Setup and Radio Link Deletion. When an NBAP request is received in the Node B, the Node Controller is responsible for assigning the received NBAP procedure to an NBAP Handler. The different ways of performing this assignment are discussed in section 4.3.1.

**Connection Manager**

This module is responsible for establishing, maintaining and releasing the SCTP connection between the Radio Network Controller and the Node B. When the Connection Manager is started it connects to the SCTP server of the Radio Network Controller. Once the connection has been set, the process delivers to the Node Controller all the messages that receives from the Radio network controller using a handler. The handler is an Erlang function that is provided by the Node Controller to the Connection Manager when it is started. The fact of using a function provided by Node Controller to perform the delivery of the received messages gives a big versatility to the Connection Manager, as it makes possible to change the way that the message are delivered without modifying the module and makes easier applying a preprocessing to the incoming messages.

To manage the SCTP connection the Connection Manager uses the `gen_sctp` Erlang module, that provides functions for communicating with SCTP sockets.

**NBAP Handler**

This module contains the functions for encoding and decoding the NBAP messages using ASN.1. It is also responsible for assigning the handling of the decoded requests to a suitable process.

To do the ASN.1 NBAP message encoding and decoding it uses the ASN.1 Erlang modules that provide functions for automatically encoding and decoding ASN.1 messages given their ASN.1 definition. The ASN.1 definition of the NBAP protocol has been extracted from [9]. The decoded ASN.1 messages are represented as Erlang records.

Once a request has been decoded, its routing is done in a different way depending on the kind of NBAP procedure. If it is a Radio Link Setup request, it is forwarded to the Cell indicated in the message. In the other hand, if it is a Radio Link Deletion, the handling of the request is assigned to the Communication Context that manages the Radio Links that have to be deleted.

A small clarification has to be done here. Although the NBAP Handler module is the one that implements the functions for encoding the NBAP messages, this task is not done by the NBAP Handler process but by the Communication Context that is responsible for building the response to a received NBAP request. In this way, the NBAP Handlers only have to perform the decoding of the incoming requests.

### 4.2.2 Channel Control Subsystem

**Cell**

The Cell module is the representation of a UTRAN Cell. There is one active Cell process for each UTRAN Cell managed by the Node B. The main tasks of this module are configuring the Cell and its common channels and handling the Radio Link Setup requests received by the Node B.

When the module is started it configures the common channels of the Cell. Once the Cell has been configured, it is involved in the handling of Radio Link Setup requests. Its role is to allocate a free Communication Context using the Context Resource Handler and giving it access
to the Transport Resource Handler and Radio Resource Handler of the card where the Cell is located.

The Cell is responsible for providing the correct Transport Resource Handler and Radio Resource Handler. This task is performed by the Cell because a Communication Context may have Radio Links that use resources of different cards, as only one Communication Context can be active for a certain UE in the whole Node B, but when a handover is taking place the UE has Radio Links in more than one UTRAN Cell. In this way, the Context does not need to be aware of the card where is located the Cell that is being used.

Context Resource Handler
The Context Resource Handler is responsible for creating and managing the Communication Contexts. When the Node B is initialized, the Context Resource Handler creates the set of Communication Contexts of its card. The number of Communication Contexts of each card is a configurable parameter. The task of the Context Resource Handler is to provide a free Communication Context to the Cell when a Radio Link Setup is being performed.

Communication Context
The Communication Context is the logical representation of a UE and manages all the Radio Links of a UE in the Node. The radio links are organized into Radio Link Sets and a Communication Context owns one or several Radio Link Sets. The Communication Context performs two operations, Radio Link Setup and Deletion.

When a Radio Link Setup is performed, the Communication Context is responsible for asking the Radio Link Set to establish the requested Radio Links and to build the response for the Radio Network Controller.

In the case of a Radio Link Deletion, it asks all its Radio Link Sets to delete the Radio Links managed by them that are contained in the request. When the Radio Links have been deleted, the Radio Link Sets that do not have any remaining Radio Link are deleted. Finally, if there is not any active Radio Link Set in the Context, it informs to the Context Resource Handler that it is free.

Radio Link Set
This module allocates, configures and releases the resources of a Radio Link Set and manages its set of Radio Links. This resources are mainly the transport bearers and the base band devices used by a Radio Link that are configured using the elements of the Operational unit.

The Operational unit at Ericsson does some computations that in the prototype are not performed. For these reason, for each Radio Link that has to be set the Radio Link Set performs a certain amount of busy work. This busy work consist in iterating over a loop. The number of iterations determines the amount of busy work performed, and it has been set in order to do 1 ms of busy work.

Radio Resource Handler
The Radio Resource Handler manages all the resources related with the radio interface. In the implemented prototype the only radio resources are the Base Band Devices.

When the module is started it creates a LINX Manager process to communicate with the Base Band Process in the Operational unit. Using the Manager it asks to the Base Band Process to inform about its radio devices. In the real system the response of the Base Band Process will
CHAPTER 4. IMPLEMENTATION

contain the characteristics of each of its devices but in the prototype the response is a list of integers, each one representing a Base Band Device.

**Transport Resource Handler**

The Transport Resource Handler is responsible for allocating and releasing transport bearers and setting transport channels for communication of user plane data with the Radio Network Controller. In the implemented prototype this part is very simplified and this module acts similarly to the Base Band Router, being only a middleware between the Radio Link Sets and the CPP. This module accepts two kind of requests, setting up and releasing a Transport Bearer. The first request expects a response form the Base Band Process that has to be forwarded to the correct Radio Link Set. As in the case of the Base Band Router, it has a dictionary that is used to forward the responses from the Base Band Process to the correct Radio Link Set.

**Base Band Router**

The Base Band Router is a middleware between the Radio Link Sets and the Base Band Process. The Base Band Router is necessary because it is not desirable to have a LINX port for each Radio Link Set, as it may be a limiting factor on the number of active Radio Link Sets. To avoid this, all the Radio Link Sets communicate with the Base Band Process through the Base Band Router. In this way, only one LINX port is needed for all the Radio Link Sets.

The main problem of this solution is that as the requests done to the Base Band Process are asynchronous, a way for forwarding the responses of the Base Band Process to the correct Radio Link Set is needed. To solve this problem each time that the Base Band Router receives a request from a Radio Link Set, it stores in a dictionary a {key, value} pair where the key is the device identifier contained in the request and the value is the PID of the Radio Link Set process that has sent the request. When a response from the Base Band Process is received in the Base Band Router, the PID of the target Radio Link Set is obtained from the dictionary using as key the device identifier contained in the response.

**LINX Manager**

The communication between the elements of the control and the operational unit is done using LINX. The LINX Manager is responsible for setting a LINX connection and sending messages between two processes.

Each process of the control unit that needs to communicate with a process of the operational unit uses a LINX Manager. In this way, LINX communication is completely hidden to the rest of Erlang processes as they only send and receive normal Erlang messages through the LINX Manager.

When the LINX Manager process is started, it receives as a parameter the name of the LINX port of the process that has to be accessed. Using this name, the LINX Manager can communicate with the target process. Once the initialization has been done, the Manager module waits for both incoming LINX messages from the target process or requests for sending LINX messages to the target process.

It does not exist a LINX module in the Erlang/OTP libraries so a linked in driver that uses the C LINX API is used to communicate using LINX from Erlang. The used driver is called Ainx and enables completely asynchronous LINX communication from Erlang with high performance.
4.2.3 Operational Unit

Base Band Process
This module simulates the Base Band of the Node B. It handles three kinds of LINX messages. One is the request of the radio resource handler to be informed about the devices of the base band. The other one is the request of setting a radio link in a device. When this message is received a new process is created that simulates the Base band device. The configuration of the radio channels is simulated by a sleep time of 10 ms. This value is not very important as it only changes the time needed to complete each request, but not the throughput of the system, that is the most important factor. When the radio channel has been configured it sends a response message. The third LINX message that this module can receive is the request of releasing a device.

CPP
This module represents the CPP and has a similar behavior to that of the Base Band process. It handles two types of requests, setting and releasing a transport bearer. When a request for setting a transport bearer is received, the CPP sends a response containing the identifier of the set bearer. When a release request is received, nothing is done.

4.3 Some Implementation Issues

In this section are discussed some implementation issues that for different reasons may have special interest.

4.3.1 Request Handling Assignment

The ASN.1 decoding and encoding of the NBAP messages requires a significant percentage of the computational resources needed to handle a request. To improve the performance of the system and make it scalable, these tasks have been distributed among several processes.

Distributing the encoding of the NBAP responses is easily achieved by assigning this job to the Communication Context that generates the response message.

To distribute the decoding of the incoming requests, many NBAP Handlers are active, and when a new NBAP message is received its decoding is assigned to one of the Handlers. In this way, the decoding of different NBAP messages can be concurrently done and the performance of the system is improved when increasing the number available of cores.

Two strategies have been considered for having multiple active NBAP Handlers: one consists on creating a new NBAP Handler process for each request (S1) and the other on having a pre-created pool of NBAP Handlers (S2).

The main benefit of the first solution is that it makes unnecessary to have a way of managing the Handlers, as when a decoder has finished its job, it automatically dies. In the other hand, in the second solution it is necessary to have a policy to choose which of the existing NBAP Handlers has to decode a certain message. The problem with the first solution is that a new process has to be created and initialized for each request and the number of active processes can be much higher and uncontrollable.

An important aspect to be considered in solution S2 is how to choose the Handler that has to decode a certain request. There have been considered two options: one is using a round robin strategy, consisting on having a queue of NBAP Handlers, assigning the decoding of a new message to the first decoder of the queue, and putting the chosen decoder to the end of the
queue (S2.Q). The second possibility consists on having a list with the free NBAP Handlers and when a request is assigned to a decoder, erase it from the list. When a decoder finishes with the decoding of a request, informs of this fact to the Node Controller and it is added to the list (S2.L). With this strategy the load balancing between the different Handlers is optimal but on the other hand the amount of messages needed is higher and it is necessary to have a way of storing the requests when there is not any free NBAP Handler.

To determine which is the best strategy the system has been tested using the three described options. For strategy S2, the number of pre-created decoders has been set to 10. In Figure 4.2 is displayed the time needed by the Node B to complete 5000 operations with a window of 200 using strategies S1, S2.Q and S2.L with different number of cores.

![Figure 4.2: Time needed to handle 10,000 operations with different request handling assignment strategies.](image)

Strategy S1 has the worst performance with all the number of cores. The results with strategies S2.Q and S2.L are almost equal and there can not be experimentally determined which is the best option.

The chosen strategy has been S2.L, which provides an optimal load distribution between the Handlers. This decision has been taken because this strategy may offer a better performance in a future development of the system, where more types of requests can be handled. In such a situation, the time needed to decode the requests can have a bigger variability and for this reason, assigning the handling of the requests using round robin, without taking into account this variability, could offer worse performance.

### 4.3.2 Distribution of NBAP Handlers when Using Multiple Cards

When the system is distributed among multiple cards, two options have been considered: distributing the NBAP Handlers among the cards or running all the Handlers on a single card.

The first option offers a better load distribution among the cards as the ASN.1 decoding of the NBAP messages is done by all the cards. On the other hand, it increases the amount of data that has to be sent between the cards.
The problem is that all the NBAP messages are received in the card that has the Connection Manager, and in order to minimize the amount of data sent between the cards, the decoding of the requests should be assigned to the card that is responsible for its handling, but to identify this card is used information contained in the request, that can not be accessed until it has been decoded. Due to this fact, with this option, in the worst case, a request will be received in the card that has the Connection Manager, send to another card to perform the decoding of the message and finally forwarded to the card which is the actual target of the request.

In the design step was decided to distribute the NBAP Handlers among the cards as it has been considered that the benefits of this option are more important than its disadvantages. To determine if this decision was correct, the system has been tested using both options. In Figure ?? is displayed the time needed by the Node B to complete 10,000 operations with a window of 200 using both strategies with different number of cards.

![NBAP Handlers Distribution](image)

Figure 4.3: Time needed to handle 10,000 operations with different number of cards distributing or not the NBAP Handlers.

As expected, the performance with the Handlers distributed among the cards is better. The time needed to complete the test is the same when only one card is used, as with only one card the Handlers can not be distributed. From this point, the difference between the two options increases, reaching its maximum when using 4 cards. In this situation the implementation with the NBAP Handlers distributed among the cards is 10% faster.

### 4.3.3 Limit of Active Communication Contexts

The Node B has a maximum number of active Communication Contexts. When the prototype is being executed in more than one card, is necessary to have a mechanism to control that the sum of active Communication Contexts of all the cards does not exceed this global limit. The problem is that the Contexts are managed locally in each card and it is not desirable to have a centralized Communication Context Controller because it could be a bottleneck of the system. The following mechanism for dynamically setting the maximum number of active Communication Contexts of each card has been implemented:
When the system is initialized, it is assigned to each card a proportional part of the Node B global limit of active communication contexts. This solution will be enough if the requests were equally distributed between the cards, but this it is not necessary true. For this reason, it may happen that a card has reached its limit of active Contexts while another one not. In this situation, it is necessary to increase the limit of active Contexts of the card that has reached its limit, but in order to maintain the consistency of the system, another card has to decrease its limit.

To make this possible, the Context Resource Handlers of all the cards are connected in a ring. If a Context Resource Handler detects that the number of active Communication Contexts in its card is close to the limit, it asks to the next Context Resource Handler in the ring to give to him the right of using a certain number N of Communication Contexts. This request contains the PID of the sender process.

When a Context Resource Handler receives this request, it checks if the original sender of the request was itself. If so, that means that any other Context Resource Handler of the ring has answered affirmatively and after waiting for a certain number of seconds, it sends the request again. If that is not the case and its number of active Contexts is below a certain limit, it answers affirmatively to the Context Resource Handler that has sent the request and decreases its limit of active Communication Contexts by N. Otherwise, the Context Resource Handler forwards the request to the next Context Resource Handler of the ring. When the sender of the requests receives an affirmative answer, increases its limit of active Communication Contexts by N.

This kind of problem, where there is a limit in the number of active resources of a certain kind for the whole Node B but not for each card could be found with other resources in the future development of the prototype, and if so this can be a very suitable solution. The main advantage of this solution is that there is no need of having a centralized controller.
Chapter 5

Performance Analysis

In this section are analyzed different aspects of the performance of the implemented prototype. The main interest is to measure the scalability of the system with different number of cores and cards. To make possible a better understanding of the performance of the prototype, the performance of some of the used protocols and tools is also tested.

5.1 Testing Tools

There have been implemented a set of Erlang modules to perform the tests and collect information about the performance of the system. The main modules used to perform the tests are the radio network controller module and the bench module, that controls the test and collects all the necessary information.

The radio network controller module simulates the behavior of a real radio network controller. When started this module initializes a SCTP server and waits until the Node B has established the connection. After that the Node B sends an Audit Required NBAP message and the radio network controller sends in response an NBAP Audit Request message. Then the Node B answers with the information of all its Cells and the radio network controller configures each of them using a NBAP Cell Setup procedure. When this have been done the system is ready to perform a benchmark.

A benchmark consists in performing a set of operations ordered by the radio network controller in the Node B. An operation is the smallest unit of a benchmark and consists in setting up and deleting a Radio Link in the Node B. This is done by sending a Radio Link Setup NBAP request, waiting for the successful response from the Node B and then sending a Radio Link Deletion request of the established link. An operation is finished when the Radio Link Deletion successful outcome is received in the radio network controller.

The speed at which the requests are sent is a determining factor of the benchmark and is controlled with the window. The window is a parameter of the benchmark that determines the number of radio links that can be simultaneously set in the Node B. This means that for each Radio Link Setup request message sent by the Radio Network Controller the window is decremented, and for each Radio Link Deletion successful outcome the window is incremented. If the window is 0 the radio network controller does not send more Radio Link Setup requests until it receives a Radio Link Deletion successful outcome. By modifying the value of the window the speed at which the requests are sent by the radio network controller is modified.
5.2 Testing Environment

The results shown in this chapter have been obtained in tests performed using two identical machines equipped with Intel Core i7-870 processors with four cores at 2.93 GHz with hyper-threading, 8 MB of L3 cache and 4 GB of RAM memory. The two machines are connected over Gigabit Ethernet through a switch. All the tests have been performed with the machines unloaded.

The fact that the used machines have hyper-threading means that they can execute 8 threads simultaneously although they only have 4 physical cores. The results obtained when using more than 4 cores are not very reliable because it is difficult to determine how well hyper-threading performs.

The usual test scenario of the prototype consists in executing in one machine the testing tools, the radio network controller module and the operational unit of each card. In the other machine is executed the Node B control unit of each card. The distribution of the different elements used for the tests is displayed in Figure 5.1.

![Diagram of testing elements distribution.](image)

Figure 5.1: Testing elements distribution.

This deploying scenario could appear to be a bit strange but it has been chosen because in this way the part that is most important to test - the control unit - has a machine for its own and is not affected by other elements. In addition, the fact of executing the two units in different computers makes that the tests reflect the delay of using LINX for communicating different computers. This is desirable because it is not clear if in a future deployment of the system using real hardware the control and the operational unit will be run in the same node or not, and deploying the prototype in this way the worst scenario is evaluated.

5.3 System Performance with Different Number of Cores

The performance of the system using different number of cores is analyzed in this section. All the tests have been done using only one card. To test the system using different number of cores, different number of Erlang Schedulers are used.

An Erlang scheduler is a thread responsible for Erlang processes scheduling and therefore the maximum number of Erlang processes of the same Erlang node that can be simultaneously in execution is equal to the number of Erlang schedulers. For this reason, the number of Erlang schedulers determines how many cores can be simultaneously used by an Erlang program.

It has to be taken into consideration that for every Radio Link Setup request received in the Node B, the prototype performs 10 ms of sleep time and 1 ms of busy work to simulate the work that should be done by the Operational layer.
There are some configuration parameters that have a great impact in the obtained results. An important factor that determines the speed-up achieved using different number of cores is the number of concurrent requests that are sent by the radio network controller. This number is mainly determined by the window. To perform this test has been used the described benchmark with 10,000 operations and a window of 200. The number of cores used - Erlang schedulers - varies from 1 to 4.

![Total Time](image)

Figure 5.2: Median time needed to complete a test consisting of 10,000 operations with different number of cores. Recall that for each Radio Link Setup request, 10 ms of sleep time and 1 ms of busy work time are performed.

The median time of ten runs with different number of cores is shown in Figure 5.2. In Table 5.1 is shown the median time needed to perform the test and the achieved speed-up in relation to using one core and the half number of cores. One of the goals of the thesis was that the system should be able to increase its performance in relation to the number of cores and the results of this test show that this goal has been achieved.

<table>
<thead>
<tr>
<th># Cores</th>
<th>Median Time (s)</th>
<th>Total Speed-up</th>
<th>Speed-up Doubling Cores</th>
<th>Operations/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12.39</td>
<td>-</td>
<td>-</td>
<td>807</td>
</tr>
<tr>
<td>2</td>
<td>6.56</td>
<td>1.89</td>
<td>1.89</td>
<td>1525</td>
</tr>
<tr>
<td>3</td>
<td>4.75</td>
<td>2.61</td>
<td>-</td>
<td>2105</td>
</tr>
<tr>
<td>4</td>
<td>3.58</td>
<td>3.46</td>
<td>1.83</td>
<td>2794</td>
</tr>
</tbody>
</table>

Table 5.1: Median time needed to complete a test consisting of 10,000 operations and achieved speed-up with different number of cores.

The obtained speed-up between using 1 and 2 cores is 1.89 and between using 2 and 4 cores is 1.83. This gives an overall speed-up between using 1 and 4 cores of 3.46. The ideal speed-up that could be achieved by doubling the computational resources is 2, but it has to be considered that the prototype also uses other resources like memory or network bandwidth, and this resources are the same independently of the number of cores, so it is normal that the achieved speed-up is below the ideal value.
Another significant value that can be extracted from the tests is the number of operations (an operation consists in setting up and deleting a Radio Link) that the system can perform in a second. When using 1 core, the system is able to handle 807 operations per second, with 2 cores 1525 operations per second and with 4 cores 2794 operations per second.

Apart from the elapsed time of the whole test, the time needed to handle each NBAP procedure in the Node B and in the Radio Network Controller has been recorded. In the Node B is recorded for each NBAP request received, the time needed to decode the message, perform the necessary actions and send the response. In the radio network controller is recorded for each procedure the time elapsed between the request is sent and the response is received. The difference between both measures is that the one obtained in the Radio Network Controller includes the time spent in sending the request and the response between both machines whereas the time recorded in the Node B not.

![Chart](image)

**Figure 5.3:** Median time needed to complete the processing of each kind of procedure in the Node B and the Radio Network Controller.

In Figure 5.3 are plotted the median time needed to handle each kind of NBAP procedure in the Node B and in the Radio Network Controller with different number of cores. These times decrease when the number of cores is increased. The reason for this is that the requests are sent in the same way from the radio network controller and in the Node B there are always the same number of processes, independently of the number of cores, so the only difference is the number of processes that can be simultaneously running. When the number of cores is increased, the processes have to wait less time to be scheduled, therefore the elapsed time needed to handle each request is reduced.

A noticeable fact is the big variability observed in the time needed to handle each request. The minimum, maximum and median time needed to complete a Radio Link Setup request in the Node B using different number of cores are plotted in Figure 5.4.
CHAPTER 5. PERFORMANCE ANALYSIS

Figure 5.4: Median time needed to handle a Radio Link Setup request in the Node B.

In table 5.2 are showed these numbers and also the standard deviation. The minimum time improves when changing from 1 to 2 cores and then it keeps almost constant. On the other hand the maximum time always improves, following a similar pattern as the median time. Taking as example the 4 cores test, the difference between the maximum time is almost 10 times the minimum. The reason for this behavior is the variability in process scheduling.

<table>
<thead>
<tr>
<th># Cores</th>
<th>Median (ms)</th>
<th>Max. (ms)</th>
<th>Min. (ms)</th>
<th>St. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>216</td>
<td>247</td>
<td>38</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>117</td>
<td>172</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>85</td>
<td>157</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>117</td>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>

Table 5.2: Time needed to handle a Radio Link Setup request in the Node B in milliseconds.

Although this big difference between the maximum and the minimum times, the St. Deviation is not very big, a 16 % of the mean in the 4 cores case. To see the distribution that follows the time needed to handle a request, in Figure 5.5 is plotted for each amount of time, the number of messages in the case of 1, 2 and 4 cores. The times follow a normal distribution, and as expected, when increasing the number of cores the center of this distribution is a lower value.
CHAPTER 5. PERFORMANCE ANALYSIS

5.4 System Performance with Different Number of Cards

Each card in the prototype consists of two Erlang Nodes, one containing the Channel Control Subsystem and a set of NBAP Handlers - also the Node Control Subsystem in the case of the controlling card - and the other one the Operational unit. To test the performance of the system with different number of cards, the same hardware is always used, but different number of Erlang nodes are created to simulate different number of cards.

All the Erlang nodes are created with only one Erlang scheduler, so each card uses one core. The distribution of the Erlang nodes among the machines is done in the same way that in the single card scenario. The Control unit nodes are located in one machine and the Operational unit nodes in the other, in this way, the performance Operational unit is not affected by other elements.

As explained in the previous section, it has to be considered that for every Radio Link Setup request, the prototype performs 10 ms of sleep time and 1 ms of busy work to simulate the work that should be done by the Operational layer.

To perform the tests has been used the described benchmark with 10,000 operations and a window of 200. The number of used cards varies from 1 to 4.

Figure 5.5: Distribution of the time needed to handle a Radio Link Setup request in the Node B.
CHAPTER 5. PERFORMANCE ANALYSIS

Figure 5.6: Median time needed to complete a test consisting of 10,000 operations with different number of cards. Recall that for each Radio Link Setup request, 10 ms of sleep time and 1 ms of busy work time are performed.

The median time of ten runs with different number of cards is shown in Figure 5.6. In Table 5.3 is shown the median time needed to perform the test and the achieved speed-up in relation to using one card and the half number of cards.

<table>
<thead>
<tr>
<th># Cards</th>
<th>Median Time (s)</th>
<th>Total Speed-up</th>
<th>Speed-up Doubling Cards</th>
<th>Operations/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12.60</td>
<td>-</td>
<td>-</td>
<td>793</td>
</tr>
<tr>
<td>2</td>
<td>7.29</td>
<td>1.73</td>
<td>1.73</td>
<td>1371</td>
</tr>
<tr>
<td>3</td>
<td>5.24</td>
<td>2.4</td>
<td>-</td>
<td>1908</td>
</tr>
<tr>
<td>4</td>
<td>4.29</td>
<td>2.93</td>
<td>1.69</td>
<td>2328</td>
</tr>
</tbody>
</table>

Table 5.3: Median time needed to complete a test consisting of 10,000 operations and achieved speed-up with different number of cards.

One of the requirements of the system was that it should be able to increase its performance in relation to the number of cards and the results of this test show that this requirement has been fulfilled.

The obtained speed-up between using 1 and 2 cards is 1.73 and between using 2 and 4 cards is 1.69. This gives and overall speed-up between using 1 and 4 cards of 2.93. The ideal speed-up that could be achieved by doubling the computational resources is 2, but the achieved speed-up is far from this ideal value. There are a few reasons that explain this fact. First of all, the tests are not performed using different machines, but by simulating them with Erlang nodes. In this situation, some of the resources like memory or network are not replicated, and for this reason, if the test has been performed using different machines the results would probably have been better. Another reason is the delay introduced by the communication between the different cards. As explained in section 3.6.2, in order to make the Node B appear as a single logical entity to the UTRAN, only one control plane connection can be set with the Radio Network Controller. Due to this fact, all NBAP messages are received in the same card, and then distributed among all the cards for its decoding. Furthermore, as it is not possible to determine which card has to handle a request until it has been decoded, it is possible that after decoding a request, it has to...
be forwarded to another card. In the best scenario, the message never has to be forwarded to another card, but in the worst situation, a single request has to be sent two times between the cards.

Another significant value that can be extracted from the tests is the number of operations (an operation consists in setting up and deleting a Radio Link) that the system can perform in a second. When using 1 card, the system is able to handle 793 operations per second, with 2 cards 1371 operations and with 4 cards 2328 operations.

Comparing the achieved performance when distributing the system among multiple cards and when using a single card but with different number of cores, the obtained results show that using more cores gives a better performance than using more cards, due to the reasons exposed above that penalize the performance of the multi-card solution.

5.5 Analysis of the Performance of the Used Protocols

In this section is analyzed the performance of the most important protocols used in the Node B prototype. This is done in order to achieve a better understanding of the performance of the system and to check if any of these used protocols can be a limiting factor in the performance of the prototype or the performance of a future development of an Erlang Node B implementation.

5.5.1 SCTP Performance

SCTP is used to communicate the simulated Radio Network Controller with the Node B. Therefore SCTP has very important effects in the performance of the system as it limits the number of requests that can be sent to the Node B. Some tests have been done to analyze the behavior of SCTP connection with different loads. To perform the test two elements have been implemented, a server and a client.

When the client is started it connects to the server and waits for receiving SCTP messages. In order to simulate a more realistic behavior, when a message is received, instead of immediately sending the response, it spawns a process that waits 10 ms and then sends the response to the server. In his turn the server waits until the client has connected and then starts sending SCTP messages to the client at the same time that waits for receiving the answers to the sent requests. There is a window to control the flow of messages sent by the server that determines how many sent messages can be pending of answer. When the window is full, the server stops sending messages until a response from the client is received. The test ends when the server has received the response of all the sent requests.

The tests have been performed executing the server and the client in different computers. The request message sent by the server consists of 100 bytes, that is approximately the size of a Radio Link Setup request, and the response sent by the client is a message of size 50 bytes, close to the size of a radio link set up successful outcome message.

Table 5.4 shows the elapsed time in the server when performing the test with 5,000 messages and a window that varies from 50 to 1600.
CHAPTER 5. PERFORMANCE ANALYSIS

<table>
<thead>
<tr>
<th>Window</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1190</td>
</tr>
<tr>
<td>100</td>
<td>730</td>
</tr>
<tr>
<td>200</td>
<td>324</td>
</tr>
<tr>
<td>400</td>
<td>213</td>
</tr>
<tr>
<td>800</td>
<td>227</td>
</tr>
<tr>
<td>1600</td>
<td>218</td>
</tr>
</tbody>
</table>

Table 5.4: Median time needed to send 5,000 SCTP messages between two machines.

The obtained results show that the time needed to send the messages improves until the window reaches a value of 400. From this point the elapsed time does not suffer any big variation. The reason for this initial improvement is that as the messages are sent more frequently, the time needed to complete the test decreases. Once the message are sent with the maximum frequency, any more improvement can be obtained by increasing the window.

The fact that when increasing the size of the window to high values the behavior of SCTP does not become worse is very important, because it means that SCTP should not be a limiting factor in the performance of the Node B as it can handle very big amounts of messages without congestion.

5.5.2 ASN.1 NBAP Encoding and Decoding Performance

Each NBAP request that is received in the Node B has to be decoded to its Erlang representation. In the other hand, the NBAP messages generated by the Node B, usually responses to NBAP requests, are created with its Erlang representation and before sending them over the net have to be encoded into a machine independent bit stream using ASN.1. This means that for each NBAP procedure received in the Node B, the request has to be decoded, and before sending the response to the Radio Network Controller, it has to be encoded. For this reason, ASN.1 performance can have a big impact in the performance of the prototype.

Two kind of tests have been done, one for testing message encoding and the other for message decoding. These tests have been performed with a single machine as it was not desirable that the results were affected by the network.

Figure 5.7 shows the time required to encode and to decode 40,000 NBAP Radio Link Setup request messages using different number of cores. The results obtained in both cases are similar, taking a bit more time the decoding than encoding of the messages. As expected the speed-up obtained in relation to the number of cores is good as the handling of one message is completely independent of the handling of the other messages.

Using 4 cores it takes a mean of 16.8 µs to encode a message and 17.3 µs to decode it. Taking as a hypothetical goal handling 5,000 NBAP procedures per second, and considering that for each NBAP procedure a message has to be decoded and encoded, the time required to decode and encode the NPAB messages will only represent a 17% of the total time.
5.5.3 LINX in Erlang Performance

The communication between the processes of the control and the operational unit is done using LINX. For each procedure carried out in the Node B, several LINX messages are sent between both units. Therefore, the performance of the LINX protocol plays an important role in the overall performance of the system.

As explained before, it doesn’t exist a LINX API for Erlang so the LINX API for C is used. To interact with the C API from the Erlang code, a driver is needed.

In order to analyze the performance of LINX in Erlang and to evaluate the penalty introduced for using a driver for interfacing the LINX API the same test has been implemented in C using the LINX API and in Erlang with the Alinx driver.

Two elements have been implemented, both in C and in Erlang, to perform the test, a server and a client.

The server waits for LINX messages, and when a message is received sends a reply containing the same data than the received message. In the other hand, the client sends messages containing a certain amount of data to the server and at the same time waits for receiving the answer of the sent messages from the server. There is a window to control the flow of messages sent by the client. This window determines how many sent messages can be pending of answer. When the window is full, the client stops sending messages until a response from the server is received. The test ends when the client has received the response to all the sent request.

The tests have been performed executing the server and the client in different machines and sending the messages between them with LINX over Ethernet, modifying the amount of data sent in each messages from 32 to 2048 bytes. Table 5.5 shows the elapsed time in the client for completing the tests consisting of 10,000 messages with a window of 200 using Erlang. It is not probable that LINX messages bigger than 128 bytes are sent between the control and the operational unit, and with this value, sending 10,000 messages only takes 220 ms, 0.22 ms per message in mean. With the obtained results, LINX communication shouldn’t be a bottleneck of
the system.

<table>
<thead>
<tr>
<th>Message size (Bytes)</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>126</td>
</tr>
<tr>
<td>64</td>
<td>129</td>
</tr>
<tr>
<td>128</td>
<td>220</td>
</tr>
<tr>
<td>256</td>
<td>321</td>
</tr>
<tr>
<td>512</td>
<td>526</td>
</tr>
<tr>
<td>1024</td>
<td>936</td>
</tr>
<tr>
<td>2048</td>
<td>1861</td>
</tr>
</tbody>
</table>

Table 5.5: Median time needed to send 10,000 messages of different sizes between two machines using LINX.

In Figure 5.8 the performance between C and Erlang implementations is compared, and as can be observed the results obtained using both programming languages are almost equal.

![Comparing LINX in Erlang and C](image)

Figure 5.8: Comparison of LINX performance from Erlang and C.

5.5.4 Comparison of LINX and Erlang Messages Performance

In the current implementation of the Node B at Ericsson, LINX is not only used to communicate between the elements of the different units, but all the inter-process communication is done using this protocol.

A set of tests have been performed in order to compare the performance of LINX and Erlang messages inter-process communication technologies. To do so, a client and a server have been implemented both in C and Erlang. In the C implementation, both processes communicate using LINX while in the Erlang implementation the communication is done using Erlang messages.

The tests have been performed in two different scenarios: running the server and the client in the same machine and running each process in a different machines. Additionally, in the Erlang
case it has also been analyzed the difference in the performance if the two processes are in the same Erlang node or not.

In Figures 5.9 and 5.10 is displayed the time needed to send 10,000 messages of sizes varying from 32 to 2048 bytes between the two processes using LINX and Erlang messages under the different scenarios described above.

![Comparing Erlang messages and LINX](image)

Figure 5.9: Time needed to send 10,000 messages with Erlang and LINX between the same and different machines.

With the obtained results, it is clear that when both processes are in different machines, the performance of Erlang messages and LINX is the same, as in this situation the determining factor is the network speed and not the protocol implementation.
The differences between both systems appear when the server and the client are in the same machine. Comparing the C implementation with the Erlang one when both processes are in the same Erlang Node shows that Erlang is in mean 2.3 times faster than LINX, independently of the size of the message, as the time needed to complete the test in this situation is constant and does not depend on the message size.

The explanation for this fact is that in this situation both protocols use some mechanism in order to share the memory between the different processes and avoid copying the data from one process to the other.

On the other hand, the Erlang implementation when both processes are run in different Erlang Nodes is slower than the one using LINX, and unlike the case when the two processes are in the same Erlang node, the time needed to complete the test increases with the size of the messages. In this scenario LINX is 3.3 times faster than Erlang when the messages are of 32 bytes and 16 times faster when the messages are of 2048 bytes.

There can be extracted some important conclusions applicable to the Node B implementation from these tests. First of all, using Erlang allows sending messages between processes of the same unit faster than with LINX, as in this situation the processes are in the same Erlang Node. The second conclusion is that if processes of different machines have to be communicated, as happens when the system is deployed among multiple cards, it does not matter which system is used because they have the same performance.
Chapter 6

Future Work

In this section is discussed the future work that can be carried out from the system developed in this master thesis. Mainly two aspects are considered: the functionalities that can be added to the prototype and different deploying scenarios of the system into real hardware.

6.1 Addition of New Functionalities

A real Node B performs a wide range of functionalities that are requested by the Radio Network Controller using NBAP procedures. These functionalities are related to radio resource management, power control and measurement report to the Radio Network Controller. The developed prototype only implements a subset of the functionalities of a real Node B, all of them related to radio resource management.

In order to have a more complete simulation of the Node B control unit much more types of NBAP procedures should be handled. It is specially important to add the functionalities related to power control and measurement report as these have not been considered in any way and may have an important impact in the design and performance of the prototype. Additionally, in order to test the prototype under more realistic scenarios and being able to simulate handovers and channel reallocations new functionalities related to radio resource management should be added.

For this reason, the handling of the NBAP procedures related to Common Channel Setup, Common and Dedicated Measurement Reporting and Radio Link Addition should be added in a second stage of development of the system.

It may also be necessary to apply modifications in the already implemented parts in order to interact with the real hardware. Especially to those parts that directly interact with the radio and transport resources, making them send the correct signals when communicating with these elements.

Additionally, it would be interesting to study the possibility of adding fault-tolerance to the system. In this way the system would be able to keep working after a software or hardware failure by taking profit of the features provided by Erlang in order to implement systems with a high degree of availability. This aspect could be especially important in a multi-card deployment scenario, as the possibility that one card fails is proportional to the number of used cards.
6.2 Possible Deployment Scenarios

In order to facilitate a future deployment of the implemented prototype in the real Node B hardware, the prototype has been developed having in mind the characteristics of the Node B hardware at Ericsson.

The Node B at Ericsson consists of a single card with one core PowerPc processor running Enea OSE, although it is expected to have multi-core machines running Linux in a near future.

![Architecture of the deployment using a Control card.](image)

Figure 6.1: Architecture of the deployment using a Control card.

There have been considered three possible deployment scenarios: A diagram of the first one is displayed in Figure 6.1. This option consists in adding a special control card connected to the Node B card where the Erlang control unit would be run, keeping the existing Node B card without modification. The second possibility is displayed in Figure 6.2. In this scenario the Erlang control unit is run in the same card that the rest of the Node B, and taking profit of some features of Enea OSE, the Erlang system is run under Linux and the other elements of the Node B under OSE. In the third option, the Erlang system and all the elements of the Node B are run under Linux. This option is displayed in Figure 6.3.

![Architecture of the deployment using a single card with OSE and Linux.](image)

Figure 6.2: Architecture of the deployment using a single card with OSE and Linux.

The first solution has the positive aspect that the Erlang system could be run in a normal multi-core computer with Linux, what will make easier and faster the deployment and testing process of the prototype controlling the real Node B hardware and using a multi-core processor.
The problem of this solution is that a lot data has to be send between both cards and the communication between them can be a bottleneck in the performance of the system.

On the other hand, the second solution does not present this problem, as the LINX communication would be done within the same card. With this solution the deployment would be more complicated - there would be necessary to run Linux and OSE in the same card - and would probably present a worse performance.

The third solution would probably present the better performance as there is less communication between cards than in the first option and all the system is run under the same operating system. The problem of this solution is that it would be necessary to do many changes in the current system to run it under Linux.

In all the proposed solutions, the Erlang control unit will interact with the elements of the real operational unit using LINX, as in the current prototype. Furthermore, in a first stage, as the prototype only implements a subset of the functionalities of the Node B, some modules of the Node B control unit implemented by Ericsson could be used to perform certain functionalities. The communication with these modules will also be done with LINX.
Chapter 7

Conclusions

The aim of this master thesis was to present a design of the Node B control unit that can handle multiple requests concurrently and scale both by the number of cores and the cards. Additionally, it was also the purpose of this thesis to determine the suitability of using a high level language such as Erlang to implement the Node B control unit. To achieve these objectives, it has been designed and implemented with Erlang a prototype of the Node B control unit that fulfills these requirements.

To make the Node B scalable, the prototype has been designed and implemented having in mind that it had to be able to handle multiple requests concurrently. To make this possible, it was necessary that the Node B did not wait to finish the handling of one request to start handling the next one. This has been achieved by dividing the handling of a request in the Node B in a series of sequential steps, each one carried out by a different functional element, following a pipelined architecture. Using this design the system can handle simultaneously as much requests as the number of steps necessary to complete its processing. To achieve an even higher degree of concurrence, when possible, the elements that are part of the pipeline have been replicated, so there can be multiple requests being simultaneously handled in each of the steps.

Furthermore, almost all the elements prototype can be replicate in different cards while maintaining the necessary conditions to make the Node B appear as a single logical entity to the UTRAN in order to be able to deploy the system on more than one card and take profit of the additional computing resources offered by these cards.

The developed prototype shows that implementing a concurrent and scalable Node B control unit with Erlang is completely feasible and the tests that have been carried out demonstrate that the performance and scalability of the system are good. Furthermore, it has not been detected any important bottleneck that does not permit achieving a high degree of scalability in the deployment of the Erlang implementation into real Node B hardware.

In the tests performed with the prototype the obtained speed-up between using one and two cores has been 1.89 and between using one and four cores 3.46. Moreover, although the system has not been tested using more cores, everything indicates that bigger speed-ups would be achieved when using more cores.

The performance and scalability of the system when distributing it among different cards has also been tested. The achieved speed-up between using one and two cards has been 1.73 and between using one and four cards 2.93. These numbers are worst than the ones achieved when testing the system with different number of cores mainly due to two reasons. The first cause is how the tests have been performed. As only two machines were available two perform the tests, to simulate different number of cards different number of Erlang nodes were created. In this
situation, some of the resources like memory or network are not replicated, and for this reason if the test has been performed using different machines the results would probably have been better. The second cause is the delay introduced by the communication between the different cards, that is not present in the single card scenario.

Apart from the design and implementation of the prototype and the results obtained with the tests, also some realistic deployment scenarios of an Erlang implementation of the Node B control unit over the real hardware used in the Radio Base Station at Ericsson have been discussed. The presented scenarios show that it is completely possible to use Erlang for implementing the Node B control unit without being necessary to do major changes in the design of the hardware of the Node B.

Another important conclusion that can be obtained from this thesis is that the way how the Node B software is developed and maintained can be improved by using Erlang. Erlang has some special features that make the design and implementation process completely different of that followed with other more commonly used programming languages like C++ or Java. The typical structure of Erlang programs, that usually consist of a set of processes that interact between them passing asynchronous messages, is much closer to the logical structure of the Node B control unit. This fact can simplify the design and implementation process of the Node B when using Erlang and may permit to reduce the time and effort needed to develop new versions of the Node B control unit.
Bibliography


[8] 3GPP TS 25.401: *UTRAN Overall Description*.


