

2. Data

2.1. Data sources

Data used in this thesis is provided by Ericsson site in Linköping, Sweden. Ericsson¹ founded by Lars Magnus Ericsson since 1876 is one of the world's leading in telecommunication industry. The company provides services, software products and infrastructure related to information and communications technology (ICT). Its head quarter is located in Stockholm, Sweden. Ericsson continuously expands its services and products beyond telecoms sector such as mobile broadband, cloud services, transportation and network design.

LTE, widely known as 4G, is the Long-Term Evolution of the 3G network. The high-level network architecture of LTE is shown in Figure 2.1 and is described as follows (Dahlman et al., 2013). The E-UTRAN, an official standard name for the radio access network of LTE, is the entire network. It handles the radio communication between the User Equipment, UE, or mobile device and the evolved base stations called eNB. Each eNB is a base station which controls and manages radio communications with multiple devices in one or more cells. Several base stations are connected to a Mobility Management Entity, MME, which handles the high-level operation of the mobile for different UEs. MME establishes a connection and runs a security application to ensure that the UE is allowed on the network. In LTE mobile network, multiple UEs connect to a single base station. A new UE performs a cell search procedure by searching for an available eNB when it first connects to the network. Then, the UE sends information about itself to establish a link between the UE and the eNB.

Ericsson makes a global *software release* in roughly 6-month cycles or two major releases per year. Each of these releases contains a bundle of features and functionality that is intended for all the customers. The software release is labeled with L followed by a number related to the year of release and a letter either A or B which generally corresponds to 1st and 2nd half of that year. Ericsson opens up a track for each software release and begins a code integration track. This track becomes the main track of the work or the focal branch for all code deliveries. There are hundreds of teams producing code and code is committed to this track continuously. In order to create a structure for this contribution, a daily *software package* is built. Every day a new software package is built which can be seen as a snapshot or a marker in

¹<https://www.ericsson.com/>

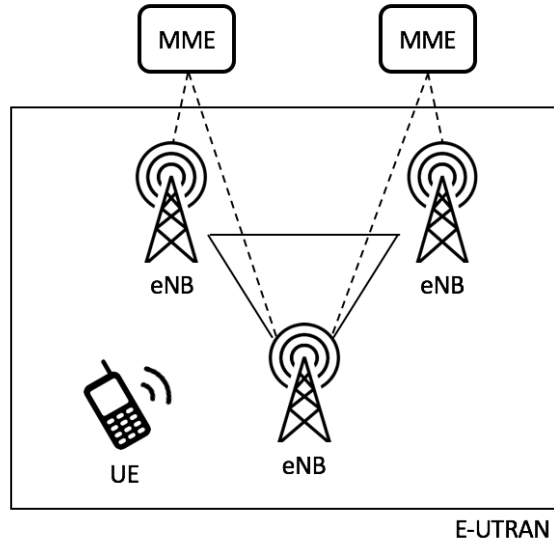


Figure 2.1.: LTE architecture overview

the continuous delivery timeline. This software package is then run through various automated test loops to ensure that there are no faults in the system. The software packages are named R followed by one or more numbers, which is then followed by one or more letters. R stands for Release-state. To summarize, each software package is the snapshot in the code integration timeline. Figure 2.2 presents the relation between software release and software package.

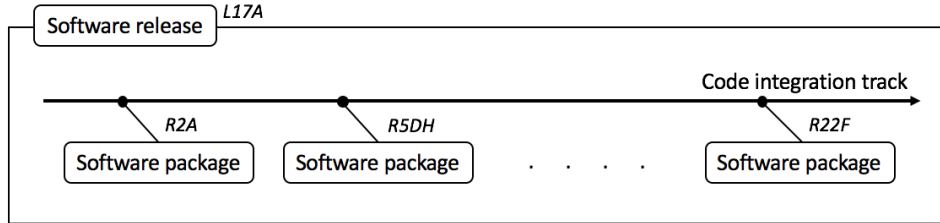


Figure 2.2.: Example of one software release that begins a code integration track. Several software packages are launched in the timeline.

There are thousands of automated tests performed. Each test belongs to a particular suite of tests, which belongs to a particular quality assurance (QA) area. For this thesis framework, one suite of test cases that test the signaling capacity within QA Capacity is focused. The QA Capacity has the responsibility of testing and tracking test cases related to LTE node capacity. Each one of these test cases has a well-

defined traffic model it tries to execute. The traffic model in this context means certain intensity (per second) of procedures, which can be seen as stimuli in the LTE node. The LTE node then increments one or more counters for each one of these procedures or stimuli it detects. These counters are called local events and they are represented by EventsPerSec.

A logging loop is started during the execution of these test cases of QA Capacity – signaling capacity. The logging loop collects a bunch of metrics, and a subset of these metrics is what this thesis is currently studied with. Once the logging loop is finished, it is written to a log file. Then, there are cron jobs that crawl through this infrastructure once a day to find latest logs and do a post-processing. The final output of these are either the CSV data or JSON encoded charts. The flowchart of this process is illustrated in Figure 2.3.

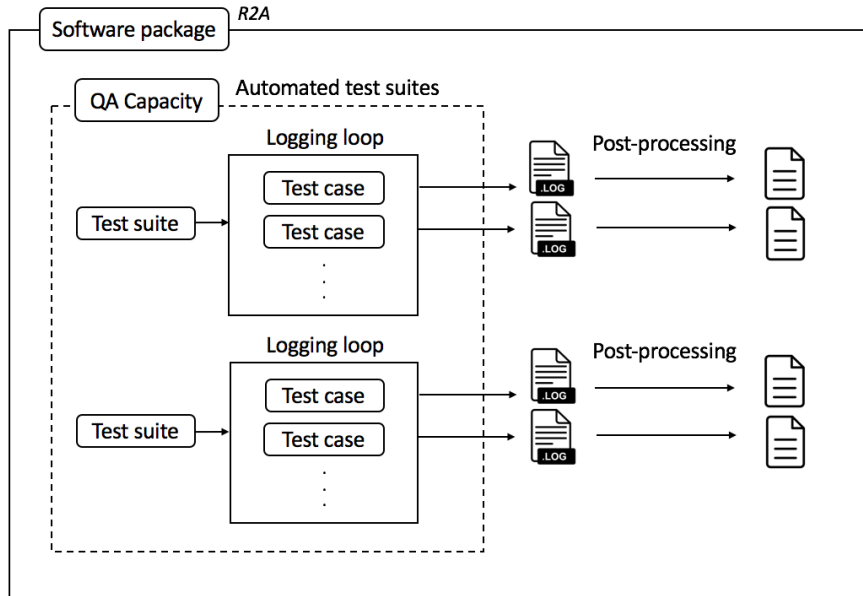


Figure 2.3.: Example of one software package. First, the QA Capacity automated test suites is started. For each test suite, a logging loop is started and a log is produced for each test case. The log file is fed to post-processing tools and the data output is obtained.

2.2. Data description

This is the data for the second generation (G2) product which contain 2,781 test cases. The data is collected on 20 January 2017 and is extracted from log file

produced by test cases. There are different types of test cases which are being executed in the automated test suites. Each test case is viewed as an observation in the data. Following are the variables in the data:

Metadata of test case

- Timestamp: Date and time when a test case is being executed (yy-dd-mm hh:mm:ss)
- NodeName: IP address or the name of a base station
- DuProdName: Product name
- Fdd/Tdd: Different standard of LTE 4G Technology. Fdd and Tdd stand for frequency division duplex and time division duplex, respectively.
- NumCells: Number of cells in the Antenna
- Release: Software release
- SW: Software package
- LogFilePath: Path for log file produced by a test case

Observable memory

- MemFreeKiB
- SwapFreeKiB
- BufferCacheKiB
- PageCacheKiB
- RealFreeKiB

CPU

- TotCpu%: Performance of a test case in terms of CPU utilization
- PerCpu%
- PerThread%
- EventsPerSec: Event intensity

This variable contains several local events that can be used when defining the test case. Apparently, there is no fixed number of local events in this variable as different test cases involve additional procedure compared to other test cases. The local events along with their values are also varied depending on which types of test cases are being executed. An example of the local events in test cases is shown in Table 2.1.

Table 2.1.: List of local events in the test cases separated by a tab character

Test case	EventsPerSec
1	ErabDrbRelease=166.11 ErabSetupInfo=166.19 PerBbUeEventTa=167.98 PerBbUetrCellEvent=12.00 ProcInitialCtxtSetup=166.20 RrcConnSetupAttempt=166.21 RrcConnectionRelease=166.11 S1InitialUeMessage=166.20 UplinkNasTransport=32.06 ...
2	ErabDrbAllocated=641.30 EventS1InitialUeMessage=142.20 McRrcConnectionRequest=142.99 McX2HandoverRequest=98.70 Paging=1399.94 PerBbLcgEvent=26.14 ...
...	...

2.3. Data preprocessing

The data consists of three software releases – L16A, L16B and L17A – and it is split into a dataset according to the software release. The dataset is sorted by the version of the software package which is named alphabetically. Even though the dataset has a timestamp when the test case is run, the timestamp is unsuitable to represent itself as a time point in time series. The software package is an important variable and more of an interest to this analysis. Therefore, it is viewed as the data points indexed in time order in the time series.

Test cases are not always being executed properly. It is either no traffic is generated during the test case or that data is not logged. If the EventsPerSec field is missing or has no value in it, the test case is being treated as incomplete and all the data related to that particular test case is ignored. This also applied for other cases when any fields in the dataset is missing.

In Table 2.1, it can be seen that EventsPerSec variable stores multiple values separated by a tab character. These tab-separated values in the field are split to columns. The process is done in order to get its local events and values which characterized the test case. These are later on used as predictor variables for fitting the Markov switching model.

Each software release consists of several software packages. Also, numerous test cases are executed in one software package. Since the software package acts as a point in time in the time series, it appears to be rather difficult to visualize all results from every executed test case for each software package. Hence, the test case which has the lowest value of the CPU utilization (or minimum value of TotCpu%) is selected to represent the performance of a specific software package. Although

taking an average of the multiple runs for test cases in the software package appears to be a good approach, it does not yield the best outcome in this case. The first reason is that manipulating data can easily be misleading. It is, therefore, settled to remain the data as it is and always visualize the data exactly as it is recorded. Another important reason for not using the average value of the CPU utilization is because the essential information in the test case will be lost. Each test case has its own local events in EventsPerSec field that used for identifying the test case. The details of these local events are absent when averaging over the CPU utilization of the test cases in the software package.

After performing all the steps describing above, the dataset of software release L16A, L16B and L17A consist of 64, 241, and 144 test cases, respectively. Lastly, each dataset with particular software release is divided into two subsets. Ninety percent of the dataset is used for training the model and the remaining ten percent is left out for testing.

Some variables in the data are chosen to be used for further analysis. There are in total one response variable and six predictor variables. Table 2.2 shows the name of variables and their description. The first three predictor variables are local events of the test case which can be found in the EventsPerSec while the last three variables are considered as the test environment. These variables appear to have a high contribution to the CPU utilization.

Table 2.2.: List of the selected variables followed by the type and the unit measure

Variable	Name	Type	Unit
Response	TotCpu%	Continuous	Percentage
Predictor	RrcConnectionSetupComplete	Continuous	Per second
	Paging	Continuous	Per second
	X2HandoverRequest	Continuous	Per second
	DuProdName	Categorical	
	Fdd/Tdd	Binary	
	NumCells	Categorical	