2. Data

2.1. Data sources

The data used in this thesis is provided by Ericsson site in Linköping, Sweden. Ericsson[1], founded by Lars Magnus Ericsson since 1876, is one of the world's leader in the 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 a 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 radio access 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 are connected 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 the 1^{st} and 2^{nd} 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 each team commit the code to this track continuously. In order to create a structure for this contribution, a daily *software package* is built which can be seen as a snapshot or a marker in the continuous delivery timeline.

[1]https://www.ericsson.com/

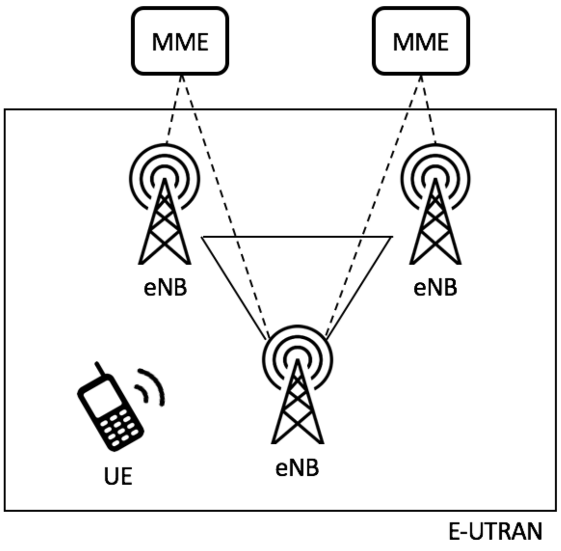

Figure 2.1.: LTE architecture overview

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,* and 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 a snapshot in the code integration timeline. Figure 2.2 presents a relationship between a software release and software packages.

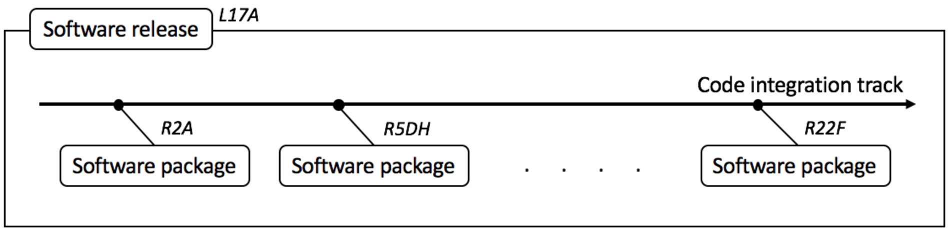

Figure 2.2.: An example of one software release that begins a code integration track. Several software packages are launched in the timeline.

There are thousands of automated tests performed. Each test belongs to a particular suite of tests, which belongs to a particular Quality Assurance (QA) Capacity. For this thesis framework, one suite of the test cases that tests the signaling capacity is focused. The QA Capacity is responsible for testing and tracking test cases related to LTE node capacity. Each one of these test cases has a well-defined traffic model that 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 that it detects. These counters are called local events and represented by EventsPerSec.

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

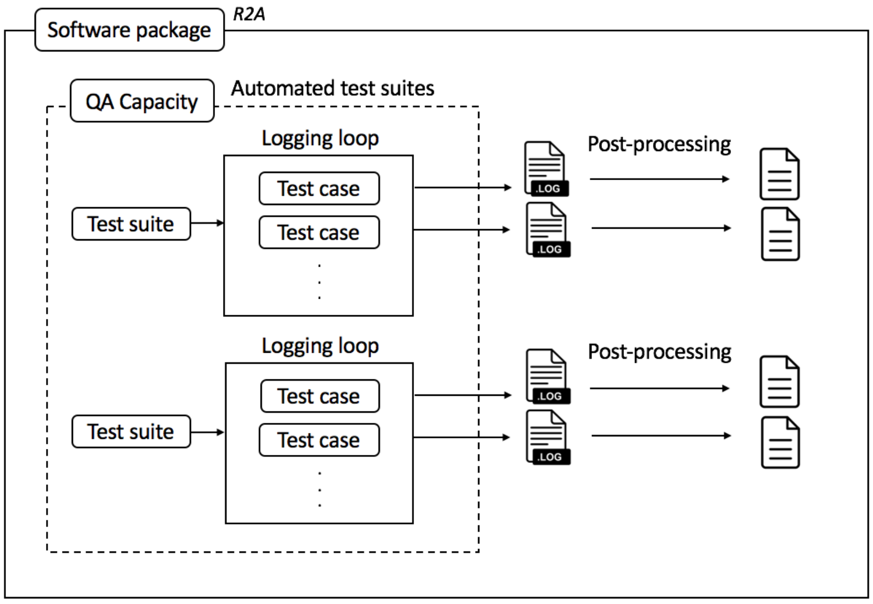

Figure 2.3.: An example of one software package. First, a QA Capacity automated test suites is started. For each test suite, a logging loop is started and a log is produced for each test case. The log file is fed to post-processing tools, and the data output is obtained.

2.2. Data description

This data is the data for the second generation (G2) product which contains 2,781 test cases. The data is collected on 20 January 2017 and is extracted from log files 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. The 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 cases. Apparently, there is no fixed number of local events in this variable as different test cases involve different testing procedures. 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, which consists of three software releases – L16A, L16B and L17A, is split into three datasets according to the software release. The test case in each dataset is sorted by its software package version, which is named alphabetically. The name of the software package is used as a time point in the time series.

Test cases are not always executed properly. The problem is either no traffic is generated during the test case or the data is not logged. This usually result in a missing value in the EventPerSec field, which causes the test case to be incomplete. The particular test case and all the data related to the test case is ignored. If the value in any other fields is missing, the test case will also be ignored.

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 into columns. The process is done in order to turn its local events and values, which characterize the test case, into a useable parameter. These parameters are later on used as predictor variables when the Markov switching model is applied.

Each software release consists of several software packages. In one software package, numerous test cases are executed. Since a software package acts as a time point in the time series, the result is rather difficult to be visualized from every executed test case for each software package. Hence, a test case that has the lowest value of the CPU utilization (or minimum value of TotCpu%) is selected to represent a performance of a specific software package. Although taking an average of 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 keep the original data and always use the unmanipulated data to visualize the time series. Another important reason for not using the average value of the CPU utilization is that the essential information in the test case could be lost. Each test case has its own local events in EventsPerSec field that is used for identifying the test case. The details of these local events will be absent if the CPU utilization of the test case is averaged.

After performing all the steps described above, the dataset of the 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 percents of the dataset is used for training the model and the remaining ten percents is left out for testing the model

Some variables in the data are chosen to be used for further analysis. In total, there are one response variable and six predictor variables. Table 2.2 shows the name of variables and their descriptions. 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 influence to the CPU utilization.

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