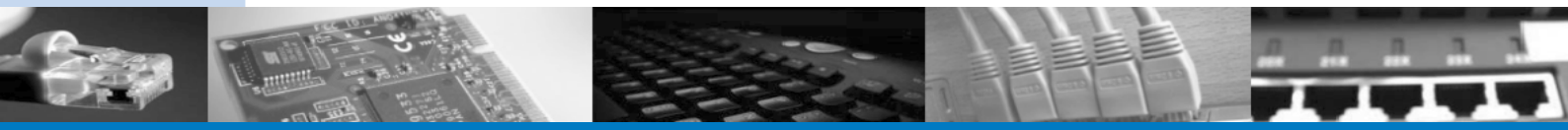
Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich

Verification of Autonomic Actions in Mobile Communication Networks

Dissertation

Tsvetko Ivanchev Tsvetkov



Network Architectures and Services NET-2017-07-1

TECHNISCHE UNIVERSITÄT MÜNCHEN
Institut für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste

Verification of Autonomic Actions in Mobile Communication Networks

Tsvetko Ivanchev Tsvetkov

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Prof. Dr. Uwe Baumgarten

Prüfer der Dissertation:

1. Prof. Dr.-Ing. Georg Carle
2. Prof. Dr. Rolf Stadler
   KTH Royal Institute of Technology, Stockholm, Sweden

Die Dissertation wurde am 27.02.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 29.06.2017 angenommen.

Cataloging-in-Publication Data

Tsvetko Ivanchev Tsvetkov
Verification of Autonomic Actions in Mobile Communication Networks
Dissertation, July 2017

Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich

ISBN 978-3-937201-56-6
DOI 10.2313/NET-2017-07-1
ISSN: 1868-2634 (print)
ISSN: 1868-2642 (electronic)

Network Architectures and Services NET-2017-07-1
Series Editor: Georg Carle, Technical University of Munich, Germany

Abstract

Mobile communication networks are highly complex systems which are comprised of a set of various technologies and automation mechanisms. Today, when we talk about cellular networks we usually think of networks that support the latest standard, which are also backwards compatible to standards developed in the past. The most prominent example is Long Term Evolution (LTE) and the supported fallback to the Global System for Mobile Communications (GSM) or the Universal Mobile Telecommunications System (UMTS). Moreover, mobile networks are comprised of a high variety of Network Elements (NEs), e.g., the LTE Radio Access Network (RAN) is typically composed of macro as well as micro and pico cells. As a result, management and automation concepts have been developed to deal with the configuration, optimization, and troubleshooting of the network. One example are Self-Organizing Networks (SONs) which aim to reduce human intervention and automatize those processes.

However, having automated entities that actively reconfigure the network raises the question of how to assess their actions and what to do in case they negatively affect the performance of the network. In the terms of SON, those entities are referred to as SON functions, also called online SON, which are implemented as closed control loops that actively monitor Performance Management (PM) / Fault Management (FM) data, and based on their objectives change Configuration Management (CM) parameters. In addition, there are offline SON methods which are comprised of sophisticated optimization algorithms that require more knowledge about the network and also more time to compute a new configuration. Usually, offline algorithms utilize simulation tools to find the most suitable configuration setup.

Nevertheless, both online and offline approaches may face difficulties while they optimize the network and may produce suboptimal or even configurations harming performance. The reasons are manifold: they may have inaccurate information about the network, they may not know whether there is another ongoing SON activity, or they may simply have a very limited view on the network. For this reason, in the mobile network area troubleshooting as well as anomaly detection and diagnosis approaches are used to analyze the network performance and state whether configuration changes had a negative impact on the NEs. Thereby, they may also suggest corrective actions that improve the current network state.

Unfortunately, such approaches often neglect issues that may occur while rolling back

configuration changes. First and foremost, verification collisions are often underestimated. They prevent two or more corrective actions from being simultaneously executed and, therefore, delay the process until the network is restored to a previous stable state. Those collisions may also result in the inability to process all corrective actions, that is, we have an over-constrained problem that has no solution for the given conditions. Moreover, the issue of detecting weak collisions, i.e., collisions that may turn out to be false positive, and SON function transactions is neglected as well.

Second, dynamic changes in the network topology, e.g., such induced by energy saving mechanisms are often neglected. They may result in incomplete cell profiles and generally complicate the process of assessing the performance impact of other configuration changes. In the worst case scenario, they may lead to the rollback of changes that are necessary for the flawless network operation.

In this thesis, a concept for verifying configuration and topology changes is presented. It is realized as three step process that splits the network into areas of interest, assesses their performance, and generates a corrective action plan. The set of corrective actions are undo actions, which restore a cell's configuration to a previous state, and topology corrective actions, which either enable or disable cells. The presented concept utilizes techniques from graph theory and constraint optimization to resolve the aforementioned issues. For example, it makes use of a Minimum Spanning Tree (MST) clustering technique that eliminates weak collisions. Furthermore, it utilizes Steiner trees to generate the necessary topology corrective actions. The presented concept is evaluated in a simulation environment and observations based on real data are made as well.

Acknowledgments

This work would not have been possible without the advice and support of many people. Here, I would like to take the opportunity to the express my thankfulness and great appreciation.

First of all, I would like to thank Prof. Dr. Georg Carle for giving me the opportunity to write my dissertation under his supervision. I am deeply grateful for his support, advice, and freedom he gave during those four years. I also would like to thank Prof. Dr. Rolf Stadler for being my second examiner and Prof. Dr. Uwe Baumgarten for heading my committee. With the same gratitude, I would like to thank Dr. Henning Sanneck for enabling me to do my research at Nokia. Our numerous discussions over the years were invaluable.

Second, I would like to thank all of my colleagues without whom I would not have been able to make it this far. Thank you, Janne Ali-Tolppa, for the various discussions, meetings, and suggestions you gave me. This thesis would not have become reality without your encouragement and your support on the ideas I had. I really enjoyed working with you. Thank you, Christian Mannweiler, for your help, support, and advice. I really do appreciate the time you always had to review my ideas and research results. Thank you, Christoph Frenzel, for your helpfulness and for constantly challenging my ideas. It was amazing to work with you side-by-side, especially during the development phase of the simulation system. I have learned a lot of things from you. Also, special thanks to Szabolcs Nováczki and Christoph Schmelz for making my entry into the research field to go as smooth as possible.

Last but not least, I would like to thank my parents, Marusya Monova and Ivancho Monov, for always been there for me whenever I needed them. You have always believed in me, and have always supported me in everything. Thank you, for your unconditional love throughout my life.

Munich, July 2017
Tsvetko Ivanchev Tsvetkov

Contents

Part I

Introduction and Background

Chapter 1

Introduction

1.1 Automation of Mobile Networks

The Self-Organizing Network (SON) concept as we know today has been developed to deal with the complex nature of standards like Long Term Evolution (LTE) and LTE-Advanced. Its purpose is to optimize the operation of the network, supervise the configuration and auto-connectivity of newly deployed Network Elements (NEs), and enable automatic fault detection and resolution [HSS11]. To be able to perform those tasks, though, a SON-enabled network has to be managed by a set of autonomous SON functions that are designed to perform specific network management tasks. Typically, they are implemented as control loops which monitor Performance Management (PM) and Fault Management (FM) data, and based on their objectives, change Configuration Management (CM) parameters.

SON functions are typically divided into three subcategories: self-configuration, self-optimization, and self-healing [3GP16g]. Within the first one fall functions like Physical Cell Identity (PCI), which is responsible for the assignment of cell IDs, and Automatic Neighbor Relation (ANR), which is accountable for the establishment of neighbor relations between cells. A representative of the second category is the Coverage and Capacity Optimization (CCO) function which has been developed to optimize the coverage within a cell by changing the transmission power or the antenna tilt. Another function that falls within the self-optimization class is Mobility Load Balancing (MLB). It has been designed to move traffic from highly loaded cells to neighbors as far as interference and coverage allows by adjusting the Cell Individual Offset (CIO) [YKK12]. The third class, i.e., self-healing, includes functions like Cell Outage Compensation (COC) which has been modeled to provide sufficient coverage by changing antenna parameters in case cells fail and induce a coverage hole.

In literature, the above-introduced functions are also referred to as online SON solutions. Nonetheless, there are approaches that fall into the so-called offline SON class. Usually, offline SON methods are supplied with PM data that has been collected for a longer period of time and, in most cases, require detailed knowledge about the net-

work. Thereby, they have the advantage of being comprised of sophisticated optimization algorithms which may provide a more accurate CM setup compared to online SON approaches. Popular examples are offline algorithms for coverage and capacity optimization which use a realistic LTE simulation scenario and create a set of CM changes by considering the outcome of the simulation runs [BFZ+14].

Nowadays, a mobile network may utilize both online and offline SON algorithms. For example, coverage changes can be made by a centralized offline algorithm whereas load balancing between cells can be achieved by using an online function like MLB. Furthermore, a network may not necessarily use all available SON functionalities. For instance, the LTE network presented in [TNSC14a] includes only two online SON algorithms, namely PCI and ANR, which fall into the self-configuration category, as introduced before.

Nonetheless, executing numerous configuration changes at the same time needs to be done with caution. Any CM change that is suggested by a SON algorithm must be coordinated in order to prevent runtime conflicts [BRS11, RSB13, ISJB14]. An example is the change of the network coverage and the adjustment of cell handover parameters. They must not happen at the same time and within the same area since the handover performance is highly dependent on how the physical cell borders are set up.

In addition, SON algorithms must be properly timed, e.g., a handover optimization algorithm like Mobility Robustness Optimization (MRO) must be triggered after CCO completes its optimization process. Otherwise, the outcome it produces may be already obsolete since the assumptions about the environment have changed.

1.2 Anomaly Detection and Configuration Restoration

Unfortunately, the increasing reliance on SON features to perform the correct optimization tasks creates a new set of challenges. In a SON, the decision to change certain CM parameters depends not only on the environment, but also on the decisions made by other SON algorithms in the past. For example, the outcome of the CCO optimization process impacts any upcoming MRO decision for the reconfigured cells. Consequently, any inappropriate CM change deployed to the network may lead to a set of suboptimal decisions in the future. In the worst case scenario, such changes may lead to network performance degradation.

There are various reasons why suboptimal configuration changes can be made. On the one hand, online SON algorithms have limited capabilities to reach the global optimum configuration. Thereby, they have a limited view on the network as well as the ongoing optimization processes. On the other hand, offline SON approaches are comprised of sophisticated algorithms that utilize tools that simulate the network and model all relevant Key Performance Indicators (KPIs). Nonetheless, such methods depend on accurate information about the network. Should there be a relatively high mismatch between simulation and reality, we may get a suboptimal configuration or even suggestions for

changes that will harm performance.

For those reasons, anomaly detection and diagnosis techniques have been studied quite extensively in the SON as well as the whole mobile network area [SN12, Nov13, CCC+14b, GNM15]. Such approaches vary to a great extent in the algorithms they utilize, in the prerequisite of having prior knowledge about the network, as well as the time they require to detect anomalies and identify the root cause. Nevertheless, they have one common task, namely to provide an accurate corrective action that will restore the performance of the network in case they detect an anomaly like an unexpected degradation in performance.

1.3 Research Objectives

The process of analyzing the impact of configuration changes on the network performance, as well as rolling back those that harm performance, requires more than simply running an anomaly detection and diagnosis algorithm. What we are actually trying to achieve is to evaluate whether or not actions that have been made in the network are complying with given requirements and conditions, as well as whether they are fulfilling the initial intents. Typically, in service and system engineering this process is referred to as *verification*, which has been also adopted by the Institute of Electrical and Electronics Engineers (IEEE) as a term that describes such a process [IEE11].

Hence, we actually have a *verification process* that assesses the performance impact of actions made in the network and provides a so-called *corrective action* in case verification fails. The assessment itself is made by using anomaly detection algorithms, however, the decision *where* and *when* to trigger verification, *how long* verification should last, as well as *which* corrective actions to deploy, induces a new set of questions which result in new challenges and issues besides those already known from anomaly detection.

The questions *"where?"*, *"when?"*, and *"how long?"* deal with the selection of the verification scope, as well as the assessment of the entities selected for verification. That is, the specification of the entities of interest (e.g., cells) as well as the choice of a strategy that decides when those entities are ready to be verified. Similarly to service and system engineering, a component that is still in a beta state or is not yet fully configured or optimized, should not be passed through verification as it will probably fail some or even all tests.

The question *"which?"* addresses not only the selection of a single corrective action, but also the ability to *identify*, *filter*, and *resolve conflicts* between sets of corrective actions. A popular example is when a cell fails verification after reconfiguring two or more of its neighbors. This results in an *uncertainty* which cell reconfiguration to blame as well as whether there is a configuration change to blame at all. Note that such uncertainties are also known as collisions. Furthermore, dynamic changes in the network topology are quite underestimated and often neglected during the process of verification. For example, if we remove cells from the network we would immediately change the

assumptions that have been initially made and may induce an anomaly.

Generally speaking, a process that verifies network changes operates differently compared to well-known SON functions. In contrast to the SON functions mentioned at the very beginning of this chapter, a verification process does not have a predefined absolute goal, e.g., minimize the number of unnecessary handovers. Instead, it forms a course of actions that have *high expected utility*, also called an action or deployment plan. In addition, it *plans under uncertainty*, either because it has incomplete information about the world, or because its corrective actions have uncertain effects on the environment. In literature, this type of planning is typically referred to as Decision-Theoretic Planning (DTP) [BDH99].

As we can see, there are numerous issues and questions that a verification strategy needs to address. This thesis is devoted to this topic and discusses four different objective categories. The very first category is dedicated to the design of a verification solution that addresses those challenges. The second one deals with issues related to the scope of verification, i.e., partitioning the mobile network into sets of cells that are independently analyzed by the verification logic. The third category is devoted to the verification of configuration changes. The fourth and last one is dedicated to the verification of dynamic topology changes.

O1: Model and design of a verification process

The process of verification in a mobile SON is a multi-step procedure that has to address questions and issues besides those known from anomaly detection. Nevertheless, before going into answering those questions we have to put the focus on specifying and designing the verification process as well as defining all the necessary properties. Hence, the first objective category of this thesis is dedicated to those topics. In particular, it is split into five sub-objectives, as follows:

O1.1 Investigate and study the disadvantages and weaknesses of approaches that verify network operations.
The objective is connected with a detailed analysis of the problems and issues experienced by current approaches, as well as the study of their weaknesses and disadvantages. Chapters 3 and 4 contribute to this topic.

O1.2 Model and design a verification process that assesses CM changes as well as changes made in the network topology.
This objective deals with the modeling and specification of the verification process. Concretely, it identifies the steps and phases that are required to enable verification in SONs as well as mobile communication networks in general. Furthermore, it characterizes the main building blocks of a verification process. Chapters 5, 7, and 9 are devoted to this topic. Note that Chapters 5 and 7 concentrate on the formal specification of the verification process whereas Chapter 9 on implementation specific details.

O1.3 Identify and specify all necessary attributes and properties of a verification process.
Here, the main tasks are to specify the corrective action types and the resulting action plan. Furthermore, it identifies the network entities that are verified and outlines the relevant verification KPI types. Chapter 5 contributes to this research objective.

O1.4 Study the relation between a verification process and concepts based on decision theory.
A verification process plans under uncertainties as well as generates a coarse of actions that have a high expected utility. Hence, it becomes of particular interest to study the connection between such a process and approaches that fall into the DTP category. Chapter 5 contributes to this objective.

O1.5 Study the relation between a verification process and approaches performing and assessing topology changes.
A verification process must also assess the impact of topology changes on the network performance. Therefore, it is of high importance to study the relation between such a process and approaches that are designed to perform and analyze topology changes. Chapter 5 discusses this topic and contributes to the objective.

O2: Verification scope selection and assessment

This objective category concentrates on the questions *"where?"*, *"when?"*, and *"how long?"*, as introduced in the very beginning of this section. That is, this category deals with the specification of the entities that are going to be verified, the trigger that starts the verification process, as well as the definition of the state which marks an entity as ready for verification. This category is split into the following sub-objectives:

O2.1 Fragmentation of the network and specifying the scope of verification.
This objective addresses the granularity of the verification operation, i.e., the fragmentation of the network into sets of network entities that will be separately assessed by a verification process. In addition, it targets the specification of the verification scope when, on the one hand, assessing configuration changes and, on the other hand, verifying topology changes. Chapter 5 contributes to this topic by specifying the general fragmentation process, whereas Chapters 6 and 7 to the selection of the scope when verifying configuration and topology changes, respectively.

O2.2 Definition of the verification state.
The aim of this objective is to specify and define the verification states of the entities that are analyzed by a verification process. Moreover, the difference between the verification states will be studied when, on the one hand, CM verification is made

and, on the other hand, when topology verification is carried out. Chapters 6 and 7 are addressing this objective.

O2.3 Estimate the duration of the verification process.
In contrast to the previous two objectives, this one is addressing the question *"how long?"*, i.e., specifying when an entity is ready for verification. As mentioned in the beginning of this section, an entity (like a cell) may fail verification if it is not yet completely optimized. Usually, this happens when a SON function requires multiple steps to achieve its optimization goal, that is, it will apply several sequential configuration changes. Chapters 6 and 7 contribute to this objective.

O3: Verifying configuration changes

The third objective category is addressing the verification of network configuration changes, also known as CM changes. In particular, it focuses on the question *"which?"*, as introduced at the beginning of this section, i.e., it concentrates on challenges related to the generation of a deployment plan of corrective rollback actions, as well as the issues that emerge because of *uncertainties* during the process of verification. This category is split into the following sub-objectives:

O3.1 Define uncertainties in the terms of a verification process.
This objective is devoted to the specification of the term "uncertainty" during the process of verifying configuration changes. Furthermore, the question when and why uncertainties emerge, as well as the question why they are considered as conflicts is targeted by this objective. Chapters 3 and 6 discuss this topic and contribute to this research task.

O3.2 Study and evaluate the impact of uncertainties on the outcome of a verification process.
The idea of this objective is to study the impact of uncertainties on the outcome of a verification process. In particular, it is observed whether the corrective action plan changes and whether uncertainties can result in inappropriate or suboptimal corrective actions. Chapter 3 contributes to the objective by studying this problem and showing why the problem emerges. Chapter 10 contributes by evaluating the impact of such uncertainties on the overall network performance.

O3.3 Resolve and eliminate uncertainties and provide accurate corrective actions when verifying configuration changes.
This objective focuses on how to resolve uncertainties and provide a deployment plan with accurate corrective actions. Furthermore, it concentrates on the detection of uncertainties which can be eliminated before generating a corrective action plan. Chapter 6 contributes to this goal whereas Chapter 10 presents the evaluation of the solution.

O3.4 Estimate the severity of the CM verification problem.
This objective aims to estimate the severity of the CM verification problem. In addition, it aims to identify the factors that influence the duration of a verification process. Chapter 6 is devoted to this research topic.

O4: Verifying dynamic topology changes

As the name of this objective category suggests, the focus is on the verification of *dynamic* topology changes, which includes the specification and generation of an appropriate corrective action. As stated in the beginning of this section, dynamic topology changes may lead to changes of the initial assumptions about the network. For example, small cells can be switched off by an energy saving mechanism if they are no longer required. Nonetheless, the disabling of a cell may induce an anomaly at the surrounding cells since they may not expect this to happen, and may therefore fail verification. Hence, we get another source of uncertainties during the process of verification.

Furthermore, the question of providing an appropriate corrective action arises. A cell is disabled when the network conditions have drastically changed. For instance, if numerous users suddenly leave the network they may cause an *unusual* or *anomalous* change in cell KPIs. Furthermore, if at the same time configuration changes are made, they can be wrongly blamed for inducing those anomalies.

In order to give an answer to those questions, this objective category is split into the following sub-objectives which aim to provide a solution to those problems.

O4.1 Specify and define the uncertainties caused by topology changes.
This objective aims to specify and show the relation between topology changes and uncertainties that emerge during the process of verification. Chapter 3 contributes to this topic by identifying the general problem, whereas Chapter 7 by further specifying the connection between the verification process and topology changes.

O4.2 Study and evaluate the impact of topology changes on the process of verifying configuration changes, as well as identify the necessary conceptual changes of a verification process.
The purpose of this objective is to study the impact of topology changes on a process that verifies configuration changes. In particular, it is of high interest to observe the consequences of those changes on the corrective action plan, on the ability to resolve uncertainties (as defined by Objective **O3.3**), and on the resulting network performance. Moreover, this objective targets the identification of the conceptual changes of a verification process that are necessary to enable the verification of topology changes. Chapters 3, 7, and 10 contribute to this objective.

O4.3 Enable topology verification, resolve uncertainties, and provide corrective actions.
This objective aims to provide a solution that enables the verification of topology changes. Concretely, it focuses on resolving uncertainties (i.e., conflicts and collisions) caused by topology changes, and the provision of topology corrective actions. Chapters 7 and 10 contribute to this research task.

1.4 Approach and Contributions

Until now, this chapter was discussing mobile communication networks, in particular, those being identified as SONs, as well as the need to verify ongoing network reconfigurations and the ability to provide appropriate corrective actions in case verification fails. Verification itself has been identified as a special type of anomaly detection that should not only be concerned about the issue of whether there is an anomaly, like a degradation in performance, but also about questions like "where to look for an anomaly?" and "how to correct it?".

The main contribution of this thesis is the introduction of a concept for the verification of such actions in mobile SONs, which targets the research objectives introduced in Section 1.3. In this document, it is referred to as "The Concept of SON Verification". The contributions of this thesis are split into four parts. The first one (cf. Section 1.4.1) is dedicated to the analysis of the problem, that is, the identification and discussion of the issues that emerge while verifying configuration changes. In addition, it contributes to the research objectives by specifying the requirements that have to be met in order to enable verification. The second one (cf. Section 1.4.2) presents the concept of SON verification which addresses the issues identified in the problem analysis part. The third one (cf. Section 1.4.3) outlines the implementation, whereas the fourth one (cf. Section 1.4.4) is devoted to the evaluation of the concept. In the following sections, an overview of those topics is given. It should be noted that the outline of the thesis and the mapping of objectives to chapters is given in Section 1.5.

1.4.1 Identification of Requirements, Issues, and Causes

The identification of the requirements that a verification strategy has to meet, needs a detailed analysis of how today's troubleshooting methods work in a SON as well as why strategies that (partially) follow the steps of a verification process fail to provide an appropriate corrective action. In particular, the focus is on the study of concepts falling within the self-healing class (cf. Section 2.3.1), as well as such that employ anomaly detection and diagnosis approaches. Chapters 3 and 4 are devoted to this study.

1.4.2 The Concept of SON Verification

In general, the concept of SON verification consists of a strategy that verifies configuration changes and a strategy that handles dynamic topology changes. In the following sections an overview is given. It should be noted that Chapters 5, 6, and 7 are dedicated to those topics.

1.4.2.1 Verification Process Analysis

The first topic that is covered here is the specification of the *verification steps*. In particular, the split of the network into sets of cell, the assessment of those sets, and the generation of a corrective action plan in the case of anomalous cell behavior.

The split of the network, also called *network fragmentation*, results into *verification areas* that include the cells that are being under observation because of configuration or topology changes. Terms and approaches from set and graph theory are used to describe and model the network fragmentation process.

The concept of SON verification further specifies the uncertainties that have been introduced in Section 1.3. It calls them *verification collisions* which result in an ambiguity during the process of generating a corrective action plan. Furthermore, a differentiation between *necessary*, *weak*, and *soft (violable)* verification collisions is made. Again, terms and concepts from graph theory are used to model and describe those properties.

Furthermore, the *verification windows* are defined, that is, the *observation* and *correction window*. They address the questions when to start the verification process, when to generate verification areas, when to trigger the algorithm that assesses their performance, and when to deploy corrective actions. To describe those phases, approaches from network management are utilized.

Finally, the *corrective action plan* is modeled by using terms known from set theory. Thereby, its properties are defined, e.g., whether the plan is *over-constrained* due to a high number of verification collisions.

1.4.2.2 Verification of Configuration Changes

The mechanism that verifies CM changes operates in three steps. Based on the CM changes it divides the network into verification areas, assesses those by using an anomaly detection algorithm, and generates corrective CM undo actions for the abnormally performing ones. Those actions restore cells to a previous stable state.

To successfully fulfill those tasks, it has to sample the network for a certain time period. However, if the mechanism is timed improperly, it may unnecessarily generate undo actions and may even prevent SON functions from reaching their set goals. To overcome this issue, for each cell a *Cell Verification State Indicator (CVSI)* is calculated. It is based on the deviation from the expected performance and is updated by using *exponential smoothing*. A verification area continuously reporting low CVSI values is considered as anomalous and processed by the verification mechanism.

However, verification areas may overlap and share anomalous cells which results in a verification collision. As a consequence, the verification mechanism is not able to simultaneously deploy some undo actions since there is an uncertainty which to execute and which to potentially omit. In such a case, it has to serialize the deployment process and resolve the collisions. This procedure, though, can be negatively impacted if weak collisions are processed, since they might delay the execution of the queued CM undo actions.

In order to overcome this issue, an approach for changing the size of the verification areas with respect to the detected collisions is developed. Concretely, it is a *Minimum Spanning Tree (MST)-based clustering approach* that is able to group similarly behaving cells together. Based on the group they have been assigned to, weak collisions are detected and cells removed from a verification area.

After completing this step, the severity of the verification problem is measured by using a technique based on *graph coloring*. It gives an estimation how many still valid collisions have remained after the elimination of the weak ones. Its outcome is used when generating the corrective action plan, which is modeled as a *constraint optimization problem* that is based on so-called *hard constraints*, i.e., constraints that must not be violated.

However, processing all planned actions may not be always possible. As stated before, verification collisions prevent two or more generated CM undo actions to be deployed at same time. As a result, the verification mechanism may not be able to process, i.e., deploy and assess, all generated CM undo actions if the time for that is limited. This gives us an *over-constrained* verification problem. To overcome this issue, a method that utilizes *constraint softening* is developed. It identifies the actions that can be merged together in order to meet the time requirement.

1.4.2.3 Verification of Topology Changes

One problem that the verification strategy, as described in the previous section has, is how it reacts to topology changes. Usually, such changes occur when cell energy saving features are activated. However, disabling or enabling cells creates uncertainties during the process of verification which may lead to an inaccurate corrective action plan or, even worse, the deployment of suboptimal actions. For example, turning on a cell may cause an anomaly at its neighbors since they did not expect a change like that to happen. In a similar manner, we are facing this problem when we do the opposite, namely turning off a cell. The neighbors may expect that the cell is always switched on during its operation. Furthermore, the fact that we need to enable or disable cells typically means that the network and service demands have drastically changed, e.g., because numerous User Equipments (UEs) have either entered or left the network. An event like that can also induce anomalies at already enabled cells for which CM verification cannot provide an appropriate corrective action.

In order to handle and verify topology changes, an approach is developed that is based on *Steiner trees*. The Steiner tree problem is a combinatorial optimization problem that tries to reduce the length of an MST by adding extra vertexes and edges to the initial edge weighted graph. Those additional vertexes are referred to as *Steiner points* whereas the initial nodes are called *Steiner terminals*. In general, Steiner points represent cells that can be turned on or off during their operation whereas terminals describe cells that remain always switched on. Based on whether a cell is used as a Steiner point to form the tree, it is decided if and how to consider it while generating the corrective action plan.

1.4.3 Implementation of the Verification Concept

The design and implementation of the verification concept is another important contribution. It focuses on the practical realization of the approaches, as introduced in the previous sections. First of all, it starts by discussing the programming language of choice as well as the packages and libraries that are required to fully implement the verification logic. An example is the constraint optimizer that is needed when verifying CM changes. In addition, packages and libraries that ease the development process and improve code readability are discussed as well.

Second, the implementation structure of the verification concept is presented. Concretely, the modules, the functions implementing the verification logic, and the different type classes are introduced. An example here is the cell clustering algorithm used the elimination of weak verification collisions.

1.4.4 Concept Evaluation

The evaluation of the verification concept is a major contribution of this thesis. It is considered as such not only because it estimates the capabilities of the introduced verification approach, but also because it studies the limits of the concept as well as investigates the impact of neglecting the issues, as discussed in Sections 1.3 and 1.4.2. It starts by analyzing the verification capabilities of already used SON features. Then, it continues by examining the consequences of neglecting uncertainties, i.e., verification collisions, while verifying network configuration changes.

The evaluation also covers the problem of having an over-constrained verification problem (cf. Section 1.4.2), as well as discusses the issue of having weak collisions. It also includes an evaluation of the strategy that defines whether the observed cells are ready for verification. Finally, it investigates the problem of having dynamic topology changes and evaluates the Steiner tree-based verification approach (cf. Section 1.4.2.3).

1.5 Thesis Outline

In this section, the outline of the thesis is presented (cf. Figure 1.1). Thereby, a short introduction of all chapters is provided as well as the mapping of the research objectives (cf. Section 1.3) to the chapters of this thesis is given.

Chapter 2: Background

This chapter provides background information about mobile communication networks, explains the basics of mobile network management, and introduces the concept of Self-Organizing Networks (SONs). Moreover, an overview of the generation of communication standards, the structure of the core and radio access network, and the standardization process is given. An introduction to the network management architecture, terminology, and SON use cases is presented as well.

Chapter 3: Problem Analysis

This chapter provides a detailed analysis of the problems that have to be solved by a verification process. Concretely, it is split into two parts. On the one hand, it discusses the challenges and issues that emerge during the process of detecting anomalies and rolling back configuration changes. On the other hand, the chapter gives a detailed description of the factors that influence this process and lead to the discussed issues.

Chapter 4: Related Work

This chapter is devoted to the work that is related to the concept of SON verification. In particular, this chapter distinguishes between three categories: pre-action analysis, post-action analysis, and post-action decision making. Representatives of the pre-action analysis class prevent conflicting changes from being executed. On the contrary, post-action analysis focuses on the assessment of already executed actions and the detection of anomalous cell behavior. Similarly, post-action decision making is devoted to the detection of suboptimal or degraded performance, however, is also responsible for the provision of corrective actions.

Chapter 5: Verification Analysis and Terminology

Verification Analysis and Terminology is devoted to the concept of SON verification. It gives an introduction to the verification terminology as well as an overview of the CM and topology verification process. Furthermore, the chapter outlines all verification-related terms, all verification properties, and the connection of SON verification to other areas, e.g., feedback planning.

Chapter 6: Verification of Configuration Changes

This chapter presents the first main building block of the concept of SON verification, namely the process responsible for the assessment of CM changes after they have been deployed to the network. Furthermore, it discusses the procedure of generating a corrective action plan as well as the strategy that is used to process the plan and maximize the total network performance.

Chapter 7: Verification of Topology Changes

In this chapter, the major topic of discussion is the process of verifying dynamic topology changes. Such changes occur when energy saving features are enabled, i.e., cells get enabled or disabled depending on the network service requirements. Furthermore, in this chapter a detailed description of the Steiner tree-based verification algorithm is given.

Chapter 8: Evaluation Environment

This chapter is devoted to the simulation environment and the real data set. They are used during the evaluation of the verification concept. It describes all main building blocks of the used simulation environment, including the network topology, the available PM data, and the offered CM parameters. It also provides a detailed overview of the data set generated by a real LTE network.

Chapter 9: Concept Implementation

In this chapter, the implementation of the verification process is discussed. In particular, the main building components are introduced, and the way of how to initialize the verification process is described. Furthermore, a description of the libraries and tools required for the implementation of the concept is provided.

Chapter 10: Evaluation Methodology and Results

This chapter is devoted to the evaluation of the SON verification concept. For the evaluation, the real LTE data set is considered as well as the simulation environment is used. Note that Chapter 8 describes those two in detail. The evaluation itself studies the concept's capabilities to verify configuration and topology changes, i.e., define the scope of verification, assess the made network operations, and generate a corrective action plan.

Chapter 11: Conclusion

This chapter summarizes the introduced concept, highlights the major findings gained during the evaluation, and gives an outlook on open questions and future work.

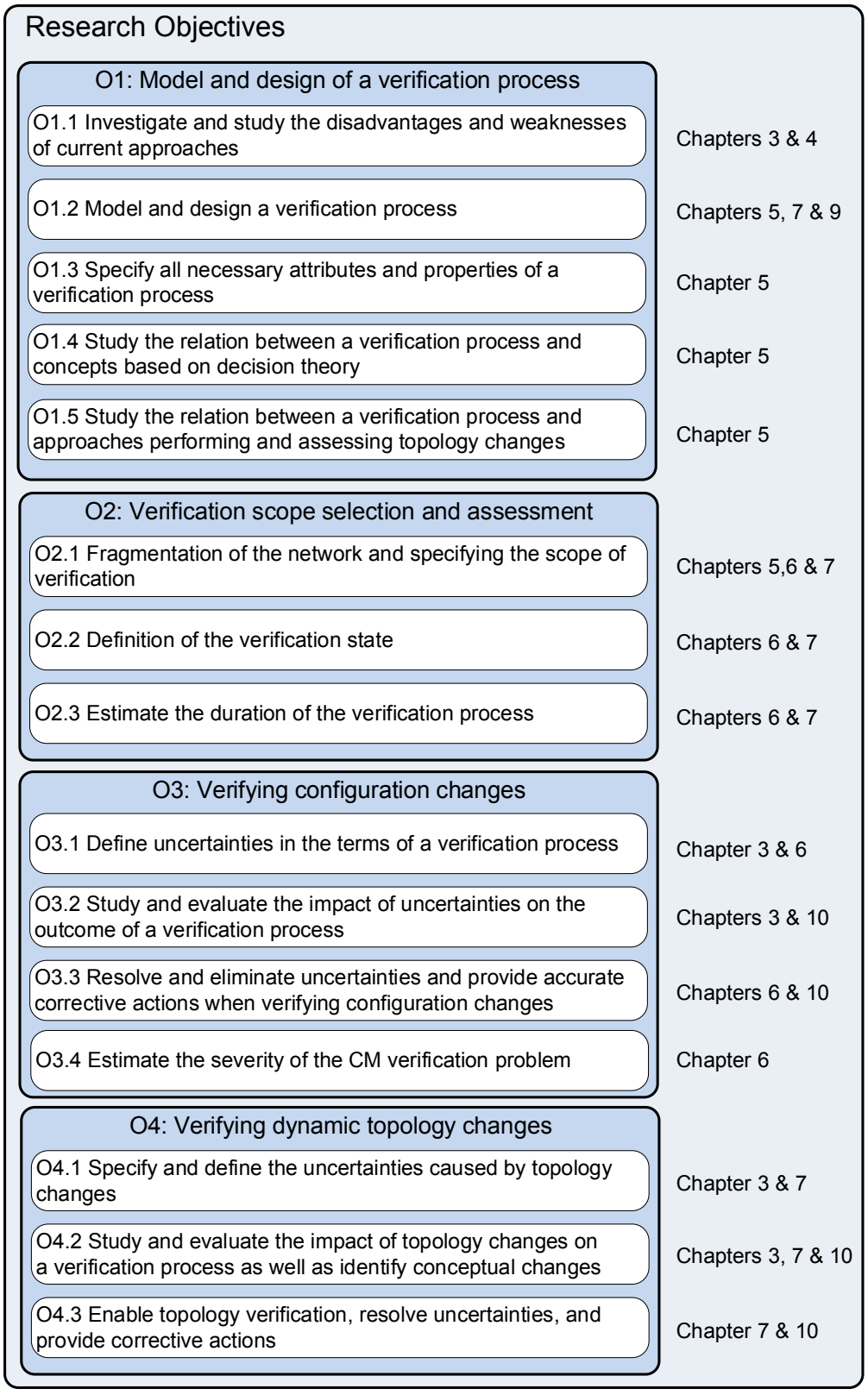


Figure 1.1: Thesis research objectives mapped to chapters

1.6 Publications

1.6.1 Publications in the Context of this Thesis

During his doctoral studies, the author has contributed to the following list of journals, conference and workshop papers, patents, as well as demonstration sessions, and has won the following awards:

Awards

- Best paper award at the IEEE/IFIP Network Operations and Management Symposium (NOMS 2016)
  - Tsvetko Tsvetkov et al., A Minimum Spanning Tree-Based Approach for Reducing Verification Collisions in Self-Organizing Networks [TATSC16a]

Journal Papers

- Tsvetko Tsvetkov, Janne Ali-Tolppa, Henning Sanneck, and Georg Carle. Verification of Configuration Management Changes in Self-Organizing Networks. In IEEE Transactions on Network and Service Management (TNSM), Volume 13, Issue 4, Pages 885-898, Invited Paper, December 2016 [TATSC16c]

Conference and Workshop Papers

- Tsvetko Tsvetkov, Janne Ali-Tolppa, Henning Sanneck, and Georg Carle. A Steiner Tree-Based Verification Approach for Handling Topology Changes in Self - Organizing Networks. In International Conference on Network and Service Management (CNSM 2016), Montreal, Canada, October 2016 [TATSC16b]
- Janne Ali-Tolppa and Tsvetko Tsvetkov. Network Element Stability Aware Method for Verifying Configuration Changes in Mobile Communication Networks. In IFIP Autonomous Infrastructure, Management and Security (AIMS 2016), Munich, Germany, June 2016 [ATT16a]
- Tsvetko Tsvetkov, Janne Ali-Tolppa, Henning Sanneck, and Georg Carle. A Minimum Spanning Tree-Based Approach for Reducing Verification Collisions in Self-Organizing Networks. In IEEE/IFIP Network Operations and Management Symposium (NOMS 2016), Istanbul, Turkey, April 2016, **Best Paper Award** [TATSC16a]
- Janne Ali-Tolppa and Tsvetko Tsvetkov. Optimistic Concurrency Control in Self-Organizing Networks Using Automatic Coordination and Verification. In IEEE/IFIP Network Operations and Management Symposium (NOMS 2016), Istanbul, Turkey, April 2016 [ATT16b]

- Tsvetko Tsvetkov and Janne Ali-Tolppa. An Adaptive Observation Window for Verifying Configuration Changes in Self-Organizing Networks. In Innovations in Clouds, Internet and Networks (ICIN 2016), Paris, France, March 2016 [TAT16]
- Tsvetko Tsvetkov, Christoph Frenzel, Henning Sanneck, and Georg Carle. A Constraint Optimization-Based Resolution of Verification Collisions in Self-Organizing Networks. In IEEE Global Communications Conference (GlobeCom 2015), San Diego, CA, USA, December 2015 [TFSC15]
- Szabolcs Nováczki, Tsvetko Tsvetkov, Henning Sanneck, and Stephen S. Mwanje. A Scoring Method for the Verification of Configuration Changes in Self-Organizing Networks. In International Conference on Mobile Networks and Management (MONAMI 2015), Santander, Spain, September 2015 [NTSM15]
- Tsvetko Tsvetkov, Henning Sanneck, and Georg Carle. A Graph Coloring Approach for Scheduling Undo Actions in Self-Organizing Networks. In IFIP/IEEE International Symposium on Integrated Network Management (IM 2015), Ottawa, Canada, May 2015 [TSC15]
- Tsvetko Tsvetkov, Szabolcs Nováczki, Henning Sanneck, and Georg Carle. A Post-Action Verification Approach for Automatic Configuration Parameter Changes in Self-Organizing Networks. In International Conference on Mobile Networks and Management (MONAMI 2014), Würzburg, Germany, September 2014 [TNSC14b]
- Tsvetko Tsvetkov, Szabolcs Nováczki, Henning Sanneck, and Georg Carle. A Configuration Management Assessment Method for SON Verification. In International Symposium on Wireless Communications Systems (ISWCS 2014), Barcelona, Spain, August 2014 [TNSC14a]

Patents

- Tsvetko Tsvetkov, Janne Ali-Tolppa, and Henning Sanneck, Method of verifying an operation of a mobile radio communication network, September 2016, WO Patent, PCT/EP2015/055786 [TATS16]
- Szabolcs Nováczki, Tsvetko Tsvetkov, and Henning Sanneck, Verification of configuration actions, March 2016, WO Patent, PCT/EP2014/069096 [NTS16]
- Henning Sanneck, Tsvetko Tsvetkov, and Szabolcs Nováczki. Verification in self-organizing networks, November 2015. WO Patent, PCT/EP2014/058837 [STN15]

Demonstration Sessions

- Tsvetko Tsvetkov, Henning Sanneck, and Georg Carle. An Experimental System for SON Verification. In 11th International Symposium on Wireless Communications Systems (ISWCS 2014), Barcelona, Spain, August 2014 [TSC14]

1.6.2 Publications in the Context of Other SON Related Areas

During his doctoral studies, the author has contributed to the following list of papers, that have been discussed in this thesis, but are not part of the presented concept.

Conference Papers

- Stephen S. Mwanje, Janne Ali-Tolppa, and Tsvetko Tsvetkov. A Framework for Cell-Association Auto Configuration of Network Functions in Cellular Networks. In 7th International Conference on Mobile Networks and Management (MONAMI 2015), Santander, Spain, September 2015 [MATTS15]

- Christoph Frenzel, Tsvetko Tsvetkov, Henning Sanneck, Bernhard Bauer, and Georg Carle. Operational Troubleshooting-enabled Coordination in Self-Organizing Networks. In 6th International Conference on Mobile Networks and Management (MONAMI 2014), Würzburg, Germany, September 2014 [FTS+14b]

- Christoph Frenzel, Tsvetko Tsvetkov, Henning Sanneck, Bernhard Bauer, and Georg Carle. Detection and Resolution of Ineffective Function Behavior in Self-Organizing Networks. In IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM 2014), Sydney, Australia, June 2014 [FTS+14a]

1.7 Statement on the Author's Contributions

This thesis is based on the journal, conference, and workshop papers in which Tsvetko Tsvetkov, the author of this thesis, is listed as first author (cf. Section 1.6.1). It should be noted that all except three papers, in which Tsvetko Tsvetkov is listed as first author, have been developed, written, and evaluated by the very same person. Only minor support was required by the co-authors. The three exceptions are:

- [TNSC14a]: the evaluation was made by Szabolcs Nováczki.

- [TSC15]: Figure 3 of the paper was supplied by Szabolcs Nováczki.

- [TATSC16b]: Figure 3 of the paper was created by Janne Ali-Tolppa.

Those parts and elements are neither considered nor used in this thesis.

Furthermore, the verification-related papers (cf. Section 1.6.1) in which Tsvetko Tsvetkov is not listed as first author are not used in thesis. Those papers utilize parts of the concept of SON verification that solve research problems that are out of the scope of this thesis. For example, SON verification can improve SON coordination decisions [ATT16b]. Nevertheless, the author of this thesis has contributed to those papers with his verification concept.

The code of the verification process (cf. Chapter 9) has been solely written by the author of this thesis. The components that establish the connection (cf. Section 8.1) with the radio simulator were written together with Christoph Frenzel.

1.8 Note on Terminology

It should be noted that throughout this thesis, several terms are used as synonyms. The following list summarizes those terms.

- Terms representing configuration changes
  - *CM changes, configuration adjustments, and (cell) reconfigurations.*
    In most cases, the term CM change is taken which is most commonly used in the field of mobile network management [HSS11, 3GP16g].
- Terms representing corrective actions
  - *Undo action, undo request, CM undo action, CM undo request, and configuration rollback.*
    Those terms represent a corrective action that returns a cell's configuration to a previous stable state. Note that the usage of the word "undo" appears at first in chapters describing the concept of SON verification since it is a term used solely by the introduced concept. In the chapters proceeding the verification concept the term "rollback" is used.
- Terms representing SON functions
  - *Online SON solution, online SON algorithm, online SON approach.*
    Those terms originate from [BFZ+14] where they are used as synonyms for SON functions, as given in [HSS11, Ban13].
- Terms related to the verification scope
  - *Rollback of the target cell of the given verification area and undo a verification area.*
    The concept of SON verification often uses the expression "undo a verification area" which is the shorter version of undoing or rolling back the configuration of the target cell (i.e., the cell of interest) of a verification area.
- Terms describing the concept of SON verification
  - *Verification approach, verification method, and the verification process.*
    It should be noted that the term "a verification process" is used in the introduction section as a generic term that represents a strategy that assesses the

ongoing network reconfigurations, and provides a corrective action if verification fails. Beginning with Chapter 5, the term "the verification process" is used as a synonym for the concept of SON verification.

Furthermore, the following remarks on the used terminology should be made:

- *Configuration changes versus topology changes.*
  In literature, it is often the case that a topology change is considered as configuration change as well. For example, turning off a cell requires the change of cell parameters like the transmission power. Nevertheless, to make a clear differentiation between reconfigurations that do not change the topology of the network and such that do so, the decision is made to use those two terms separately.

1.9 Document Structure

This section summarizes the main elements used in this document to introduce, for example, an important term, or provide a listing of references on which the given chapter is based on.

1.9.1 Reference to Author's Publications

The chapters that are based on papers published by the author of this thesis are listed at the beginning of a chapter. They are formated in the following way:

> **Published work** This box list published papers on which a certain chapter is based on and which are part of the presented concept.

It should be noted that Section 1.7 lists the author's contribution.

1.9.2 Terminology and Notes

Important terms are introduced as definitions, as given by the example below. All definitions can be found in Chapter 3 (Problem Analysis), and Chapters 5 to 7 which are devoted to the concept of SON verification.

Definition 1.1 (Exemplary definition)**.** A definition gives in most cases a formal specification of a problem or terminology that play a crucial role in the concept of SON verification.

1.9.3 Objectives and Tasks

As shown in Section 1.3, the tasks of this thesis are defined by objective categories and objectives. An objective category should be seen as a generic thesis task which is comprised of a set of several other tasks that are given by the objectives. An example is shown example below.

O1: First thesis objective category

Description of the objective category.

O1.1 **First thesis objective.**
Description of the first objective.

O1.2 **Second thesis objective.**
Description of the second objective.

Furthermore, there are evaluation tasks that can be found only in Chapter 10. They can be identified as follows:

- ✓ First evaluation task
- ✓ Second evaluation task

They list the topics and tasks that are going to be discussed and evaluated in the given case study.

1.9.4 Summary and Findings

All chapters conclude with a summary. In addition, a research objective (cf. Section 1.3) is met by giving an answer to one or more research questions, which are highlighted as shown below. The chapters that contribute to research objectives are explicitly listing those answers in the summary section.

O1.1: Name of the research objective
Q: First research question that contributes to the objective
A: Answer to the first question and description of the findings.

Q: Second research question that contributes to the objective
A: Answer to the second question and description of the findings.

1.9.5 Appendix

The appendix, which can be found at the very end of this thesis, lists the used acronyms, symbols, figures, tables, listings, definitions, and cited references. It should be kept in mind that this thesis makes extensive use of various symbols and acronyms, which further increases the importance of the first two lists.

Chapter 2

Background

This chapter provides an introduction to communication standard generations, the network and protocol architecture, as well as the collaboration and standardization processes. Section 2.1 is dedicated to those topics. Furthermore, relevant topics from the mobile network management and SON area are discussed. In particular, the management architecture, terminology, and data sets are presented, as well as an overview of the SON categories and the structure of SON functions is given. Sections 2.2 and 2.3 discuss those topics in detail. Finally, Section 2.4 concludes the chapter with a short summary.

2.1 Mobile Communication Networks

2.1.1 Generations of Communication Standards

The development of mobile telecommunication systems started in 1980s with the introduction of the first generation of mobile communications, also abbreviated as 1G. It consists of analog telecommunication standards like the Nordic Mobile Telephone (NMT), which has been widely used in the Nordic countries, Switzerland, and Eastern Europe, and Advanced Mobile Phone System (AMPS), which was widely adopted in North America and Australia. They were replaced by digital communication technologies, also known as 2G standards, in the early 1990s. Today, the most popular 2G standard is the Global System for Mobile Communications (GSM) which has been developed and standardized by the European Telecommunications Standards Institute (ETSI). The primary objective of GSM was to allow users to roam throughout Europe and offer services compatible to those known from Integrated Services Digital Network (ISDN) standards [Sch03].

The 2G communication systems evolved over time. In the year 2000, the General Packet Radio Service (GPRS) was introduced which was a packet-oriented service that allowed higher data rates compared to what GSM could initially reach. GPRS was accompanied by the Enhanced Data Rates for GSM Evolution (EDGE) which was introduced in the year 2003. It introduced a new modulation scheme and higher data rates.

The successor, namely communication systems of the third generation (3G), emerged in the late 1990s. One of the most popular representatives is Universal Mobile Telecom-

munications System (UMTS) which has been standardized by the 3rd Generation Partnership Project (3GPP). It makes use of Wideband Code Division Multiple Access (WCDMA) and provides a greater bandwidth and spectral efficiency. In addition, it simplifies the network architecture including the Radio Access Network (RAN), the core network, as well as the UE authentication procedure.

The fourth generation of mobile communication systems, also abbreviated as 4G, emerged in the year 2008. The most popular representative is LTE which was developed with the intention to meet the following requirements [STB11]:

- Reduce connection & transmission delays, and cost per bit.
- Increase user data rates and cell-edge bit rate.
- Greater flexibility of using the spectrum, and greater automation of the system.
- Simplified network architecture.

LTE itself is specified through releases. The first one is release 8 which reached a sufficient level of maturity in December 2007 and, as a result, was considered as a new Radio Access Technology (RAT) of the International Mobile Telecommunications (IMT) family. In the meantime, LTE was further developed which led to release 9. It addressed the applicability of the technology in other regions, in particular North America, as well as introduced improved positioning methods and support for new broadcast modes. In addition, it defined new requirements for lower power nodes, e.g., pico base stations.

The next release, also labeled as release 10, was the starting point for LTE-Advanced [HT12]. The main improvements included carrier aggregation that increases the total transmission bandwidth, as well as improvements in uplink and downlink Multiple-Input and Multiple-Output (MIMO). Furthermore, support for relaying and cell interference coordination was added.

2.1.2 Collaboration and Standardization Process

The collaborative specification model was initially introduced with the GSM system and became de facto the development model for UMTS and LTE [STB11]. This resulted in a world-wide collaboration scheme that includes not only the ETSI, but also development and standardization organizations from North America, China, South Korea, and Japan. Thereby, the 3GPP was founded in 2011 and included 380 partner companies. 3GPP itself is divided into three technical specification groups [3GP16a]:

- Group responsible for the RAN, in particular, the definitions of the functions requirements and interfaces.
- Group responsible for service & system aspects, i.e., the overall architecture and service capabilities.

- Group responsible for the core network and terminals, i.e., specifying terminal interfaces and capabilities, as well as defining the core part of 3GPP systems.

Furthermore, each of those groups is further split into working groups, each having a specific responsibility. Their task is to prepare, maintain and approve Technical Specifications and Technical Reports, also abbreviated as TS and TR, respectively. They are used by the project partners, e.g., incorporate them into deliverables or standards. Each report or specification has its unique number which identifies to which series (subcategory) it belongs to. For instance, the 32.500 series covers SONs.

2.1.3 Radio Access Network

The Radio Access Network (RAN) is a crucial part of a mobile telecommunication system since it implements the radio access technology and provides to UEs access to the core network. Each communication standard (cf. Section 2.1.1) defines its own RAN. For example, in GSM the RAN is called GSM Radio Access Network (GRAN) or GSM EDGE Radio Access Network (GERAN) in case the network also implements EDGE features. In UMTS, the RAN is referred to as an Universal Terrestrial Radio Access Network (UTRAN), whereas in LTE it is called an Evolved Universal Terrestrial Radio Access Network (EUTRAN).

Since the technology that is used in this thesis is LTE, also because of the fact that SON features firstly appeared in LTE [HSS11], this section will concentrate on the EUTRAN.

2.1.3.1 EUTRAN Overview

Let us start with Figure 2.1 which shows the overall EUTRAN architecture according to [LL08, HSS11, 3GP16b]. It is comprised of a set of Evolved NodeBs (eNBs) which are connected with each other over the X2 interface and connected to the core network over the S1 interface. In the latter case, the eNBs are connected to the serving gateway / Mobility Management Entity (MME) of the core network.

Furthermore, the protocols that are used in the RAN are called Access Stratum protocols whose tasks can be summarized as follows [STB11]:

- Radio resource management: includes all functionalities related to the mobility control, the allocation of resources to UEs, as well as all other radio-related functions.
- Evolved Packet Core (EPC) connectivity: provides the signaling towards the packet core.
- Header compression: offers an efficient use of radio resources, e.g., by compressing Internet Protocol (IP) packet headers.
- Positioning of UEs: specifies and provides all the required data that determines a UE's position.

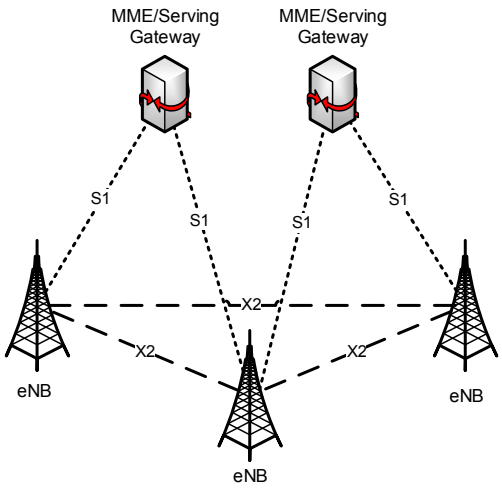


Figure 2.1: Structure of the EUTRAN according to [3GP16b]

- Security of the transmitted data: provides encryption methods for the data transmission over the radio interface.

In addition, the functions that implement those features reside in the eNBs, i.e., the control in the EUTRAN is realized in a distributed way.

2.1.3.2 Handover Procedures

In LTE we can distinguish between idle mobility mode and connected mobility mode. Note that those modes are also referred to as Radio Resource Control (RRC) idle and RRC connected [HT09]. In RRC idle, cell re-selection is made autonomously by the UE and is based on the measurements the UE has collected. Moreover, the cell re-selection is made according to the parameters that are broadcasted by the network which is very similar to the approach used in WCDMA and High Speed Packet Access (HSPA) networks. This procedure is also known as cell selection or cell camping in the Public Land Mobile Network (PLMN). In particular, each UE receives the broadcast channels, estimates the radio link quality and determines the most suitable cell candidate. After the completion of this phase, the UE registers itself at the selected PLMN.

In RRC connected mode the handover is controlled solely by the network and is based on the activity and movements of the UE itself. Concretely, the RRC connection is controlled by the eNB which makes use of Radio Resource Management (RRM) algorithms to perform this procedure. Those algorithms operate on layer 1 to 3 of the eNB user plane and control plane protocol architecture (cf. Section 2.1.5). They take into account parameters like the Allocation Retention Priority (ARP), Channel Quality Indicator (CQI), the uplink and downlink Guaranteed Bit Rate (GBR), and the QoS Class Identifier (QCI).

In addition to those modes, we can distinguish between intra-RAT and inter-RAT handover. Within the first category fall handover procedures that manage the UE mobility within the same LTE RAT. They can be further divided into inter-frequency and intra-frequency handover which, as the name suggests, handle the mobility between cells operating at the same or at different frequencies. The inter-RAT handover is responsible for the mobility between different technologies like LTE and WCDMA/HSPA.

2.1.4 Core Network

The core network in LTE, also referred to as EPC, is accountable for the control of the UEs and the establishment of all necessary connections, e.g., to networks of older generations. Figure 2.2 visualizes the simplified structure, including the main nodes and interfaces.

The MME is the node which handles all the signaling between the UEs and the core network [STB11]. In particular, it is responsible for the allocation and release of resources based on the activity of the UEs [HSS11]. It also handles the interworking with other legacy networks which is established over the Serving GPRS Support Node (SGSN), as shown in the same figure. The protocols it runs are known as Non-Access Stratum protocols. In addition, it is communicating with the Home Subscriber Server (HSS) which holds a user's subscription data, roaming restrictions, Quality of Service (QoS) profile, as well as information that allows a user to connect to the Packet Data Network (PDN).

The primary function of the next node, the serving gateway, is to act as a user plane tunnel, i.e., all packets are forwarded and routed through it. Moreover, it is a mobility anchor for data bearers in case UEs move from one eNB to the other [STB11]. It is also accountable for administrative tasks like the volume of data a user has sent or received which is typically required for charging purposes.

The responsibility of the last node, the PDN gateway, include IP address allocation, filtering of user IP packets based on their QoS profile, and policy enforcement. It is also responsible for the QoS enforcement of GBR bearers. The PDN gateway is also the edge router of the whole evolved packet system.

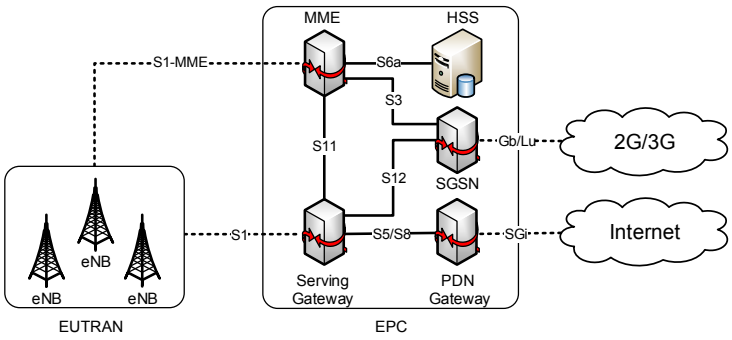


Figure 2.2: Overview of the EPC according to [3GP16b]

2.1.5 Protocol Architecture

The radio protocol architecture of the EUTRAN is divided into two protocol stacks: user plane and control plane. In this section, a brief introduction to each of those protocol stacks is given. It is based on [STB11, HSS11].

2.1.5.1 User Plane

The user plane protocol stack (LTE layer 2) consists of three sub-layers which implement the following functionalities:

- Packet Data Convergence Protocol (PDCP) layer: the main functions of this layer are header compression, retransmissions during handover (cf. Section 2.1.3.2), and integrity protection & ciphering.
- Radio Link Control (RLC) layer: the main responsibilities of the RLC layer are packet segmentation, reassembly, and reordering. Segmentation is required for data transmission over the radio interface.
- Medium Access Control (MAC) layer: the main function of the MAC layer is the multiplexing of data from two or more radio bearers. It is also able to negotiate a QoS level for each bearer.

2.1.5.2 Control Plane

The functionalities of the LTE control plane can be split into procedures that handle cell (re-)selection procedures when UEs are in idle mode and the RRC protocol in the case UEs are in connected mode (cf. Section 2.1.3.2). The responsibilities of the latter one include the RRC connection control which, as the name suggests, is covering the setup and release of an RRC connection. Furthermore, it handles the reporting of measurements required for the handover of UEs, the transfer of UE radio access capability information and the broadcast of system information messages.

The cell (re-)selection procedures handle the measurement rules, e.g., when to start to search for a new cell, the frequency or RAT evaluation, the ranking of a cell, determining the cell access restrictions and the verification of a cell's accessibility.

2.2 Mobile Network Management

As we saw in the previous sections, mobile communication networks are highly complex systems. As a result, their maintenance and management becomes a difficult task, mainly because of their configuration interdependencies, the number of elements they are comprised of, as well as the necessity of being backwards compatible. The backwards compatibility is required since a mobile network is usually comprised of communication

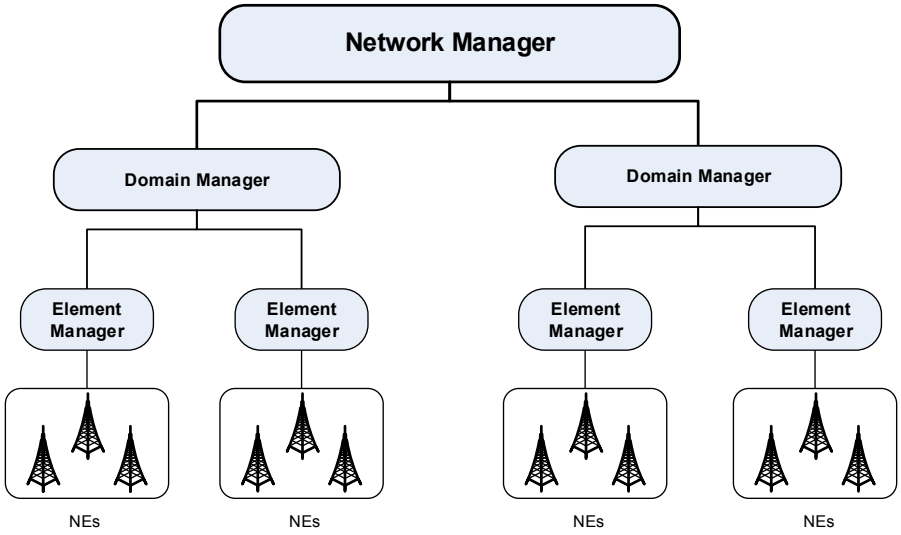


Figure 2.3: Overview of the OAM architecture

technologies of different generations, e.g., GSM, UMTS, and LTE (cf. Sections 2.1.3 and 2.1.4).

Mobile network management is typically identified as a centralized Operation, Administration and Management (OAM) architecture [HSS11]. It makes use of optimization and planning tools that are semi or fully automated and may, in addition to that, require human supervision. The used management mechanisms operate based on PM and FM data that gets exported by the network, and suggest or autonomously deploy CM changes.

In the upcoming sections, an overview to the architecture is given, and a description of the data types that are relevant for network management is provided.

2.2.1 Operation, Administration and Management Architecture

The OAM architecture is a hierarchical system that is comprised of three management levels. In literature, they are referred to as the Network Management (NM), the Domain Management (DM), and the NE level [HSS11]. The controllers operating at those levels are typically called the network, domain, and element manager. The latter one is also abbreviated as EM. Figure 2.3 visualizes the OAM architecture.

At the bottom we have the NEs, e.g., eNBs, which are controlled by the corresponding EMs. One level up, i.e., the DM level, we have the domain manager which manages the EMs. The communication itself usually requires proprietary interfaces. Furthermore, the entities that fall below the same domain manager represent a domain that is provided by a single vendor. The communication to other domains, however, is established over open interfaces. The manager at the topmost level, i.e., NM, manages the entities of the DM level. The communication is made over proprietary or standardized interfaces.

There are advantages and disadvantages of where an entity, like a SON function

(cf. Section 2.3.2), is located. At the NE level it has an immediate and quick access to all ongoing activities, as well as direct access to the most recent PM parameters. However, at that level an entity will also have a limited view on the network, e.g., it does not have information about the activities occurring in another domain. In order to get a wider view, it would need to reside at the DM or even NM level. Being there, though, induces delays until it finally manages to get the relevant data. For example, it may take up to several minutes until it collects the necessary PM data. Hence, an entity may have a coarse view on the ongoing activities.

2.2.2 Network Management Data

As stated above, the data that is relevant for network management tasks is split into PM, FM, and CM. In this section, a detailed description of those data types is given. Note that the observations below originate from [3GP01, 3GP13a, 3GP16e].

2.2.2.1 PM Data

There are three main classes of indicators that are widely used in mobile network management:

- Performance counters: such counters represents a single value that is maintained and used by the NE, e.g., a variable that counts the fault events that have been spotted by the NE.
- Key Performance Indicators (KPIs): a KPI is a well-known (standardized) formula that uses one or more counters as input and computes its outcome according to the formula definition [3GP16f]. Examples can be found in Section 8.1.1.2.
- Key Quality Indicators (KQIs): a KQI typically aggregates two or more KPIs in order to provide a high level view on the network performance. However, because of this property they are not suitable for the assessment of a cell's performance [HSS11].

2.2.2.2 FM Data

In general, FM data is represented by alarms that are generated by the NE and sent to the OAM system. An alarm message is a predefined event which falls within one of the following categories:

- Hardware failures: occurs when physical resources of an NE are malfunctioning.
- Software faults: appear, for instance, when a software upgrade has failed or when database inconsistencies emerge.
- Functional failures: happens when an NE looses its functional resources which are not due to hardware issues.

- Capability loss: occurs when an NE is no longer able to provide its services, e.g., due to overload.
- Communication failures: alarms that fall within this class are generated when the communication between two NEs or an NE and the operations system is disturbed.

2.2.2.3 CM Data

In the terms of network management, CM data can be split into two main categories:

- Passive CM: passive CM data offers information about the made changes, i.e., it can be seen as a configuration overview report or even as a configuration logging mechanism.
- Active CM: active CM data offers the operator or any other automated entity to actively change parameters of the physical NE.

Passive CM data is usually fetched by sending a request, e.g., from the NM level. The data itself contains information about the time when the object was created or deleted, as well as a listing of the parameters that were actually changed. The real data set that is used in this thesis (cf. Section 8.2.1.2) contains of such passive CM data reports.

Active CM data on the other side allows not only the retrieval of the current CM parameter settings, but also enables their adjustment. Examples are the transmission power or the antenna tilt degree of a cell. The simulation environment that is described later in this thesis (cf. Section 8.1.1.3) offers access to such CM data.

2.2.3 Granularity Period

A granularity period is a term describing the frequency at which network performance data is collected by a measurement job [HSS11]. At the end of a granularity period a report is generated for the NE of interest which includes the measured types and resources. In [3GP16e], the data collection methods are further being discussed, however, they are beyond the scope of this thesis.

The 3GPP document also mentions that each NE has to retain the measured data until the report has been sent to, or retrieved by the corresponding Operations Support System (OSS) destination database. It is also possible to buffer reports at the corresponding Element Manager (EM). However, the storage capacity and validity of the report is implementation specific.

2.3 Self-Organizing Networks

The Self-Organizing Network (SON) concept [HSS11, RH12, HT12] as we know today has been developed to deal with the complex nature of standards like LTE and LTE-Advanced (cf. Section 2.1.1). It has been introduced to optimize the operation of the

network, supervise the configuration and auto-connectivity of deployed NEs, and enable automatic fault detection and resolution. The first SON features have been used for managing the mobile RAN [NGN08b].

In order to perform those tasks, though, such a network has to be managed by a set of autonomous entities, called SON functions, that are designed to perform specific network management tasks. As shown in Figure 2.4, they are implemented as closed control loops which monitor PM and FM data, and depending on their objectives, change CM parameters. Furthermore, a SON function's goal is given by the operator through a function's configuration [3GP13b]. One example is the MRO function which tries to minimize the call drops, radio link failures as well as unnecessary handovers by adjusting the CIO parameter [3GP16b]. Another example is the MLB function that is designed to move traffic from overloaded cells to neighbors as far as interference and coverage allows [YKK12].

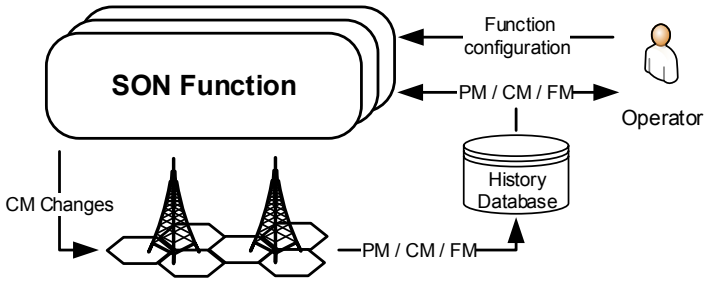


Figure 2.4: Overview of a SON

2.3.1 SON Categories

Usually, SONs are split into three main categories: self-configuration, self-optimization, and self-healing. They are the major topic of discussion in this section.

2.3.1.1 Self-Configuration

Self-configuration, as the name suggests, deals with the roll-out and the auto-configuration of NEs while minimizing at the same time human intervention. As mentioned in [SBT10a, SBT10b, HSS11], self-configuration is a process that consists of three phases.

First and foremost, after the installation of a base station the so-called auto-connectivity setup is triggered. It has the purpose of establishing the connection between an NE and the DM system it has been added to. In particular, it is responsible for the basic NE connectivity, like connecting the Dynamic Host Configuration Protocol (DHCP) server and setting up the initial IP address. During this phase, the certificates required for the secure communication with the remaining NEs as well as the initial configurations are downloaded.

Second, a transition to the auto-commissioning phase is made. During this phase, the installed software is validated, i.e., a comparison between the current software version and version required for the particular NE type is made. If required, the installed software can be updated. Furthermore, the configuration data is activated by rebooting the NE.

Third, the dynamic radio configuration phase is triggered. It completes the processes stated during the auto-commissioning phase: it adapts the NE to the current network deployment and adjusts the radio parameters accordingly. Thereby, the PCI, the neighbor relations, the initial antenna tilt and transmission power settings are configured. This step is supposed to replace planning tool sets which are typically required in the presence of legacy base stations.

2.3.1.2 Self-Optimization

Self-optimization is also seen as a step beyond self-configuration. The outcome of the initial configuration process yields a set of parameter values which may become suboptimal or even obsolete over time. The most common reasons for this to happen are environmental changes, which may lead to a change of the propagation conditions, and dynamic changes of the network service demands. In the latter case, the most common trigger are changes in the traffic behavior as well as user movements. The objective of all self-optimization mechanisms is to monitor the network, detect suboptimal configurations and update them accordingly.

In general, self-optimization approaches can be further divided into such that put their focus on the mobility of UEs, such that concentrate on optimizing the cell coverage and the Random Access Channel (RACH), and such that actively change the network topology. Typical representatives of the mobility optimization category are MRO and MLB. One the main objectives of MRO is to improve the handover performance between two neighboring cells. Note that a detailed description of MRO is provided in Section 8.1.2.1. The MLB function on the other side tries to offload a cell by distributing UEs to its neighbors. Such a distribution procedure is available for both idle and connected mode (cf. Section 2.1.3.2). In LTE, typical KPIs that are considered by this function are Reference Signal Received Power (RSRP) and Reference Signal Received Quality (RSRQ). Furthermore, since release 8 it is possible to exchange load information over the X2 interface which can be also taken into account [HT09].

A function that is optimizing the cell coverage is CCO. There are three parameters which are controlled by this function: the antenna azimuth, the transmission power, and the antenna tilt degree. Popular KPIs considered by CCO are the CQI and the Signal to Interference plus Noise Ratio (SINR). Note that the CCO function is further discussed in Sections 8.1.2.2 and 8.1.2.3.

A function that falls within the topology change category is Energy Saving Management (ESM). Its goal is to reduce energy consumption in the network by turning off cells that are no longer necessary while guaranteeing that network coverage as well as user perception do not suffer. In addition, there are numerous triggers that can activate such a

function. It can be a fixed time schedule, but it can be also triggered by unusual load or traffic data.

2.3.1.3 Self-Healing

The third category, self-healing, is responsible for fault detection and resolution [3GP16g]. In general, a self-healing process consists of two parts: a monitoring and a healing process part. During the monitoring part, the network is actively observed and the search for a so-called trigger for a self-healing condition is made. Concretely, the trigger can be hardware or software faults, configuration and planning faults, and issues that emerge due to unexpected environmental changes.

During the healing process part, the necessary performance and configuration data is fed into a diagnosis component. It can be implemented as a simplistic rule based system which is actively supplied with expert knowledge. It is also possible to utilize sophisticated algorithms, for example, such that make use of Bayesian networks [BKKS16] or neural-based algorithms [LRL+05].

After completing this step, the result is evaluated and a notification is sent to the operator. A potential recovery action, like a fallback to a previous stable state or the correction of a certain CM parameter may be suggested.

A well-known representative of the self-healing category is the COC function. As discussed in [HSS11], the reasons for cell outage can be hardware or software failures. As a result, there might be an unexpected loss of coverage and, therefore, any type of service offered by the affected cell can be lost. The detection itself is based on the PM and FM data, including all relevant UE logs. The recovery action can be a simplistic cell restart or the change of the neighbors' transmission power or antenna tilt.

2.3.2 SON Functions

The core functionalities of SON, as introduced in Section 2.3.1, cannot be implemented without the use of SON functions. In this section, their structure and activity schemes are discussed. In addition, SON management and coordination paradigms are outlined.

2.3.2.1 Structure and Phases

In general, a SON function operates in three logical phases: monitoring, algorithm, and action phase. During the monitoring phase a SON function observes certain KPIs and collects information about the network such as CM changes and fault occurrences. Note that there are different scheme types (cf. Section 2.3.2.2) which define when a function is becoming active.

After gathering the required information, the algorithm part of a SON function can get triggered. Its purpose is to compute new CM parameters which will be applied during the action phase. Here, it should be noted that a SON function can be stateful as well as stateless. For example, the algorithm part of an MRO function may change the CIO only

based on the current handover KPIs, like the handover drop rate or handover to wrong cell rate (cf. Section 8.1.2.1), without considering past KPI values. Another example is the CCO function which may track the impact of its changes on the network performance and even correct them if necessary (cf. Figure 3.2, Section 3.2.1).

2.3.2.2 Activity Schemes

Within the SON area, we can distinguish between three activity schemes. The first one is the on demand scheme which allows a SON function to become active only when it receives an explicit triggering event. Troubleshooting approaches are known to use such a scheme since their activity is required in the presence of critical alarms (cf. Section 2.2.2.2). The second scheme is the timed scheme whose primary goal is to trigger the SON function on time interval basis. For instance, an ESM function may become active only in the late evening in order to optimize energy consumption. The third scheme is known as a continuous scheme which requires the SON function to actively monitor the network. Typically, representatives of the self-optimization category use this scheme, e.g., MRO continuously observes the handover performance between neighboring cells.

2.3.2.3 Coordination and Management

Since SON functions may perform changes to network configuration parameters during their operation, a coordination entity is required to reject requests that would lead to conflicts and allow those which would guarantee a flawless network operation [HSS11, BRS11, ISJB14]. Every time a SON function decides to change a network parameter, it contacts the SON coordinator by sending a CM change request. The latter one acknowledges the change only if there has not been another conflicting function activity for the given impact area and time. In literature, this type of coordination is usually referred to as *pre-action SON coordination* and is based on rules used to avoid known conflicts between SON functions.

In literature, there are numerous suggestions of how to categorize function conflicts [KAB+10, CGT+11, JFG+13, Ban13]. One popular grouping is the differentiation between configuration, measurement, and characteristic conflicts. Configuration conflicts occur when SON functions operate on shared CM parameters. For example, two functions change the same handover parameter and are, therefore, in a conflict. Measurement conflicts appear when the activity of one function influences the input data used by another one. Characteristic conflicts usually emerge when there is a logical dependency between the active functions.

In a SON, every function comes with two essential properties required for coordination: the impact area and the impact time. As stated in [Ban13], a SON function instance has to be considered by a SON coordinator for the whole time period it has an impact on the network. This includes not only the delay required to take the measurements, run the

algorithm and compute new configuration parameters, but also the time needed to deploy the new configurations and the time until they become visible in the PM data.

The impact area on the other side is the spatial scope within which a SON function modifies configuration parameters or takes its measurements from. Concretely, it consists of the function area, i.e., the set of cells that are configured by the function, the input area, i.e., set of cells where the function takes its measurements from, the effect area, i.e., the set of cells that are possibly affected by reconfiguration process, and the safety margin which provides a higher degree of protection against undesired effects by extending the effect area.

2.3.3 Other Application Areas

Besides the RAN of a mobile telecommunication system, SON concepts have been developed for other parts of a mobile network, e.g., the core network. Also, SON mechanisms have started to emerge in other networks, e.g., Wi-Fi networks. In the following sections, an overview to those SON approaches is given.

2.3.3.1 SON for Core Networks

The up to now introduced SON principles are also applicable to the management of core networks (cf. Section 2.1.4) [NGN08a, 4GA11, HSS11]. SON functionalities that have been defined for the core network can be split into such handling the packet core and into mechanisms responsible for the voice core. Within the packet core mechanisms fall functions responsible for the auto-configuration of core elements, e.g., the auto-configuration with S1 setup [3GP16c] which significantly reduces the work required to manually configure MMEs and eNBs. Also, mechanisms that provide automatic software updates and upgrades are falling within the same class. Typically, they regularly check the repository server for a new software version and also provide post-condition checks that determine whether an element is operating as intended.

In addition to auto-configuration, so-called MME pooling and signal optimization functionalities can be utilized by the core network. The first one implements features that select the less loaded MME from a pool of MMEs [3GP16d]. Also, it provides an option for smart offloading of subscribes without interrupting the active session. As for signal optimization, features like paging load reduction are provided. Usually, the load generated by such messages increases as the number of eNBs grows. Therefore, this SON functionality allows the selection of the most suitable paging strategy, e.g., to page at first the eNB to which the UE was most recently connected to.

Within the pool of voice core SON mechanisms fall Voice over IP (VoIP) quality monitoring and management features. First, such features collect data from the network and create an end-to-end quality view of VoIP calls that are taking place. As mentioned earlier in this chapter, performance data can be collected from the element itself or from a centralized location. Second, a VoIP SON mechanism is attempting to improve the

utilization of network resources, for example, by providing alternative traffic routes.

Even though there are SON mechanisms specified for the core network, there are also such that consider the RAN, i.e., they are responsible for the functionality of both networks at the same time. A prominent example is the configuration transfer procedure that allows the exchange of information between two eNBs over the MME [3GP16d]. A concrete application is the exchange of the IP addresses of the eNBs, which are required to establish X2 links [3GP16b]. Such links are usually created by the ANR function (cf. Section 8.2.2.2) which is responsible for the automatic establishment of neighbor relations.

2.3.3.2 Wi-Fi SON

Another area that is utilizing SON features is Wi-Fi, also referred to as Wi-Fi SON [All17]. In general, Wi-Fi SON features aim to simplify indoor Wi-Fi networking by minimizing human intervention [Qua16]. The idea is to ease the management of wireless routers, smart gateways, access points, and range extenders. The hardware equipment itself may be provided by different vendors. Similarly to mobile SON, Wi-Fi SON defines separate key categories. They can be summarized as follows:

- Self-configuration: allows the auto-configuration and plug-and-play deployment of Wi-Fi SON devices. This category is very much alike mobile self-configuration, as introduced in Section 2.3.1.1.

- Self-managing: deals with the optimization of the monitored network elements. It is the equivalent of the self-optimization area that is already known from mobile SONs (cf. Section 2.3.1.2).

- Self-healing: it is responsible for the detection and the remedy of wireless connectivity issues and bottlenecks, i.e., it has the objectives already known from the mobile self-healing SON category (cf. Section 2.3.1.3).

- Self-defending: prevents and secures the wireless network from unauthorized access. It should be noted that this category is not explicitly defined for mobile SONs. Security features are usually incorporated in self-configuration mechanisms.

A common example for Wi-Fi SON is an indoor setup which is covered by several routers. They typically support several standards, like 802.11a, b, g, n, ac, and come in addition to that with multiple frequency support, e.g., 2.4 GHz and 5 GHz [IEE16]. Self-configuration enables the automatic device setup, e.g., frequency choice, with minimal or even without human intervention. Self-optimization and self-healing provide the network, for instance, with multiple connection paths by using the available range extenders. Self-defending is responsible for detecting suspicious behavior within the network, identifying the root cause, and blocking the responsible entity.

Furthermore, there are SON functionalities that are responsible for the concurrent optimization and management of Wi-Fi and mobile networks. One of the most common examples is multi-layer LTE & Wi-Fi traffic steering [LAB+13]. It studies the ability to steer users between Wi-Fi access points and LTE bases stations based on the dynamic behavior of the network. The goal is to offload cellular traffic as well as to make Wi-Fi access points an integrated piece of the network provided by an operator. Otherwise, without the provision of such a SON capability both the mobile and the Wi-Fi network can easily become congested which, as a result, may lead to the degradation of service quality and user experience. The KPIs based on which steering is carried out are the network load, the experienced QoS, and the radio link quality.

In literature, there are numerous approaches that implement as well as analyze the impact of such steering functionalities [NPS11,BCS11,PKVB11,LLY+13]. Typically, the suggested algorithms differ in the specific goals they are following, as well as the metrics and criteria used during the decision making process. In general, they can implement only a simplistic threshold comparison, e.g., if the Received Signal Strength (RSS) exceeds a predefined limit, or make use of additional parameters like time-to-trigger or hysteresis values.

2.4 Summary

In this chapter, an introduction to mobile communication networks is provided. First, it starts by introducing the generations of communication standards and the most popular representative technologies, i.e., GSM, UMTS, and LTE. The latter one is discussed in detail since it represents the technology that is used in this thesis. Second, the chapter presents the basics of mobile network management, including the OAM architecture and the relevant data types, that is, PM, FM, and CM data. The process of data acquisition from the network is provided as well. Third, the SON concept is described, in particular, SON categorization, coordination, and management paradigms. In addition to that, different SON application areas are discussed, in particular, SON for the RAN and core of mobile networks as well as SON for Wi-Fi networks.

Part II

Problem Analysis and Related Work

Chapter 3

Problem Analysis

In this chapter, the challenges and issues that emerge while verifying network operations are discussed. First of all, Section 3.1 motivates the need for a CM assessment strategy, which is also referred to as a verification process. In particular, it gives an overview on troubleshooting and anomaly detection approaches as well as the problems that may occur due to automatic CM changes. Further, the section highlights paradigms for self-organized networking and also puts the focus on the term "verification".

Second, in Section 3.2 the problems are discussed that are faced by a verification process. Concretely, it focuses on the issue of interrupting SON function transactions as well as the presence of so-called verification collisions. The latter ones lead so several other issues which are discussed throughout the section. In addition, the problems induced by dynamic topology changes are described in detail.

Third, Section 3.3 outlines the factors that negatively impact a verification process. More precisely, the availability of PM and CM data, the statistical relevance of PM data, the characteristics of CM changes, as well as the properties of the network itself are presented. In the latter case, the density and heterogeneous nature of mobile communication networks are discussed. Finally, Section 3.4 summarizes this chapter and lists the major scientific contributions.

Published work This chapter originates from the work made in already published papers. In [TNSC14a], the term "SON verification" is introduced and an overview to already existing strategies is presented. In [TNSC14b], verification collisions are introduced and the problem is researched. Paper [TSC15] further researches the verification collision problem as well as highlights the factors that influence a verification process. Papers [TFSC15] and [TATSC16a] describe the issues that are induced by verification collisions. Paper [TAT16] focuses on the SON function transaction problem whereas paper [TATSC16b] on the problems caused by dynamic topology changes. Paper [TATSC16c] presents a summary of the problems that emerge during CM verification as well as the factors impacting the process. It should be noted that this chapter analyzes those topics in much more detail.

3.1 Motivation

3.1.1 Troubleshooting and Anomaly Detection

The maintenance as well as the process of troubleshooting a mobile network has become a complex task. The main reasons for that are the size of today's cellular networks, their heterogeneous nature, as well as the high variety of configuration capabilities. As a result, problems can appear in all areas of a cellular network, nevertheless, one of the most critical domains is the RAN [HSS11]. Active base stations are not only responsible for the coverage within dedicated areas, but also for the UEs using their services. However, there is also almost no redundancy which means that if the performance of a base station suddenly decreases, users will start experiencing a drop of QoS since the reliability and availability of the network drops as well.

In order to detect base stations that are not properly fulfilling their tasks, anomaly detection and diagnosis techniques are most commonly used. Generally speaking, an anomaly is understood as "something that deviates from what is standard, normal, or expected" [Oxf05]. In most cases, though, the main focus is on detecting abnormal behavior, for instance, whether the performance of a cell has notably degraded. Figure 3.1 shows the generic structure of an anomaly detection and diagnosis procedure. During the

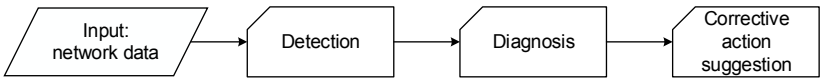


Figure 3.1: Generic overview of anomaly detection and diagnosis approaches

detection part the network is scanned for abnormal cell behavior. Thereby, the network is typically profiled [Nov13, CCC+14b], i.e., performance indicators are analyzed and the expected network behavior is specified. Usually, there is more than one profile, for example, there could be one that specifies the usual weekday behavior and another that defines how cells should perform during the weekend. Then, based on those profiles it is determined whether the performance of the cells is considerably different from the one that is expected. Should this be the case, the identification of the possible root cause is triggered and potentially a corrective action is generated.

In today's mobile networks, there are numerous reasons why anomalies actually appear. As stated in [HSS11], there are four major sources that can lead to abnormal cell behavior:

- Hardware problems
- Software problems
- Environmental changes
- Network planning and configuration changes

Within the first category fall problems related to the physical NEs. Usually, such problems are corrected by resetting or replacing the faulty element. The second category includes problems that occur after updating or upgrading the software of an NE. A typical corrective action is to restore the software to a previous point that ensures the normal operation of the NE. The third category usually includes changes that disturb the radio, e.g., demolition or construction of buildings. An expected reaction would be to do re-planning or trigger a coverage optimization mechanism. Within the last category fall CM changes that have harmed network performance, which are also the most challenging ones to detect and correct.

There are numerous reasons why it is possible to deploy suboptimal CM changes. First of all, we have to take a closer look of how mobile networks, in particular, how SONs are being managed and optimized. In general, we can differentiate between online and offline optimization [BFZ+14]. An offline SON method needs to wait until all the needed PM data is collected from the network. Then, an optimization algorithm is triggered, e.g., one that optimizes the coverage and capacity of a cell. Finally, the suggested CM changes are applied. In addition, one of the major advantages of such offline SON solutions are the sophisticated optimization algorithms they utilize. Hence, they can test a high number of CM settings and search the complete optimization space. However, those algorithms also require detailed knowledge about the network, e.g., the user locations or the received signal strength for all possible antenna tilt degrees. This type of information is often difficult to collect which may result in inaccurate assumptions about the network and the deployment of suboptimal CM settings.

Online SON solutions, also known as SON functions, determine, collect, and measure the KPIs of interest as well as deploy CM parameter changes by themselves. Those changes are based on past KPI and CM parameter values [GJCT04, AJLS11]. Furthermore, they are executed in a coordinated manner [RSB13], e.g., the concurrent optimization of coverage and handover parameters of a cell is not permitted since it is considered as a configuration conflict [Ban13].

In contrast to offline methods, online SON techniques do not require simulation models of the network, which means that they cannot be influenced by inaccuracies of the used models. However, they have the disadvantage of only optimizing the network locally, that is, they change parameters based only on the information that has been gathered around the monitored cell or base station. For example, an MRO function is usually only interested in the handover behavior of a pair of neighboring cells.

Moreover, SON functions operate without the knowledge of other upcoming SON activities. For instance, a CCO function does not know whether an MRO function will become active in the future, e.g., after CCO has finished its optimization process. As a consequence, any suboptimal decision made by one function in the present may result in suboptimal changes made by other SON functions in the future. This is also why online SON algorithms must avoid large CM parameter changes since they may lead to unexpected performance drops, e.g., a low QoS level [BFZ+14].

The corrective action for SON activities is the rollback of the configuration changes that led to anomalous cell behavior. Strictly speaking, the corrective action should be just one, namely a rollback of all configuration parameters that did harm the network performance in any way. However, this is only possible if we have a sophisticated diagnosis component which is able to provide us with that action. Unfortunately, in a mobile communication network feeding such a component with accurate knowledge is a challenging task. As mentioned in [SN12], such networks are manifold, generate different types of PM data, and react in a different way to the same corrective action. In addition, in most cases it is impossible to reuse the results from one study for the diagnosis on a different RAT, even within the same mobile network. The speed at which such systems evolve also plays a negative role since it makes many diagnosis results obsolete and unusable.

Furthermore, generating just one single corrective rollback action presumes that we know in advance the *combined performance impact* of each CM change combination. Obviously, this is a complex task as a real mobile network consists of a high number of CM parameters. For instance, the LTE network described in Section 8.2 consists of 141 configuration parameters. As a result, we cannot test every possible parameter combination and foresee all possible anomalies that may occur after a configuration change as well as do not know the impact of their rollback.

In such a case, the solution is to *sequentially* deploy rollback actions by starting with those that would *most likely* restore the network performance. Hence, we do not only observe the impact of the initially executed CM changes on the performance, but also assess the impact of the deployed corrective rollback actions. However, this type of behavior gives us what we would categorize as *a verification process*.

Definition 3.1 (Verification process)**.** For a set of deployed configuration actions $\{\delta_1, \ldots, \delta_i\}$, a verification process observes the impact of each action δ_i on the network and assembles, if required, a set of corrective rollback actions $C^{\perp}$ that revert configuration changes to a previous stable state. After processing $\forall c^{\perp} \in C^{\perp}$, the performance of the network is returned to a stable state.

However, finding and executing corrective actions is a challenging task as we are going to see in Section 3.2. Furthermore, while carrying out the assessment of the corrective actions we have to monitor CM changes that have been executed in the meantime. This means that we are not only assessing the rollbacks but also other newly made reconfigurations.

3.1.2 The Term "Verification"

The idea of evaluating whether a certain system or service complies with its requirements and targets has been known for quite a long time. For instance, in the field of software engineering verification is known as the process of determining whether the developed software can fulfill all expected requirements. This is a complex process that may involve

the use of formal methods for proving or disproving the correctness of the implementation and may, in addition to that, consist of extensive tests that examine the behavior of the software at execution time.

Within the area of mobile communication networks and, in particular the SON field, the term verification is widely being used. For example, in self-configuration the purpose of site build verification is to detect problems that may occur during the installation or integration process of new NEs [Eri12]. It may also include shake-down tests that check the network's reliability and accessibility. In the field of self-optimization, SON functions may track their own CM changes and correct their actions if they have moved away from achieving their objective. Even in the area of SON coordination, the decision to acknowledge or reject a function's request can be based on past experiences, i.e., a SON coordinator is not only designed for conflict prevention and resolution, but also verifies whether its decisions have a positive impact on the network performance. Such an approach has been introduced in [ISJB14], where reinforcement learning is used to improve the coordination between the MRO and MLB function.

As stated in the previous section, the term verification is to be seen as a superordinate concept to anomaly detection strategies. It does not only try to determine whether there is an anomaly in the network, but also to rollback actions that have caused anomalous behavior and resolve the potential conflicts when rolling back configuration changes. Thereby, it operates under uncertainty, tries to maximize the performance of the network by restoring cell configurations while minimizing the number of rollback actions. Note that a description of those uncertainties is provided in Section 3.2.2.

3.2 Challenges for a Verification Process

In the previous section, we saw how troubleshooting and anomaly detection are carried out in SONs, as well as an introduction to the verification process was given. Here, we are going to continue with this process, however, we will start by describing the challenges it faces and the issues that may occur. Each of the following sections describes a different challenge or problem.

3.2.1 SON Function Transactions

Today, SON functions usually require more than one step until they finally manage to reach their goal. One example is the CCO function [HSS11, 3GP14b] whose objective is to provide sufficient coverage and capacity in the whole network area by adjusting the antenna tilt or the transmission power. Figure 3.2 shows the simplified flow-chart of the algorithm.

It starts by collecting PM and CM data, continues by determining the statistical relevance of the gathered PM data, and triggers an antenna configuration change for the cell experiencing a problem. Furthermore, it observes the impact of its last deployed antenna tilt or transmission power change, and corrects that if required. As a result, it reaches

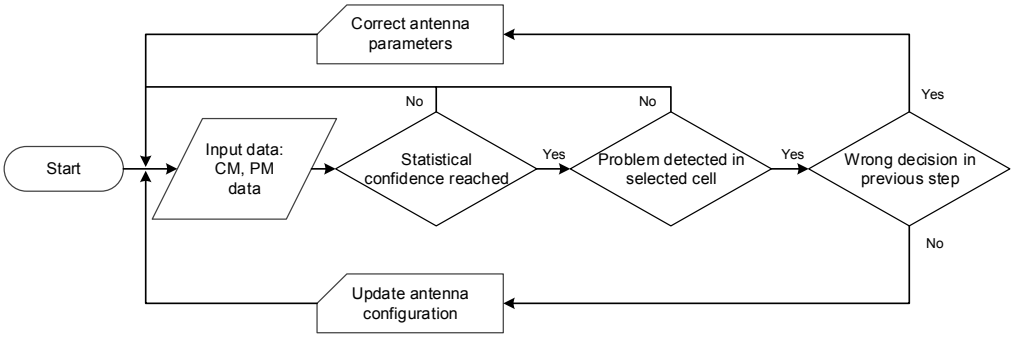


Figure 3.2: Simplified flow-chart of the CCO algorithm

its goal after executing a set of actions δ, which is also referred to as a *SON function transaction*.

Definition 3.2 (SON function transaction)**.** A SON function transaction $\Delta = \{\delta_1, \ldots, \delta_i\}$ is a set of actions $\delta_1, \ldots, \delta_i$ executed by a function that are required for reaching its optimization goal. An action δ corresponds to a CM change or a set of CM changes.

However, during a transaction a SON function may induce temporal performance drops in the network. Let us give an example. Suppose that in order to achieve its objective, a SON function needs to execute two actions (cf. Figure 3.3). The first action triggers a change within cell 1 whereas the second one adjusts a parameter within cell 2. Now, suppose that after executing the first action a performance drop occurs at both cells. If we immediately rollback the first action, we will interrupt the transaction and prevent a SON function from achieving its objective.

Furthermore, if a second function is triggered after the completion of this transaction, it may create a set of suboptimal changes since the initial ones that were required were

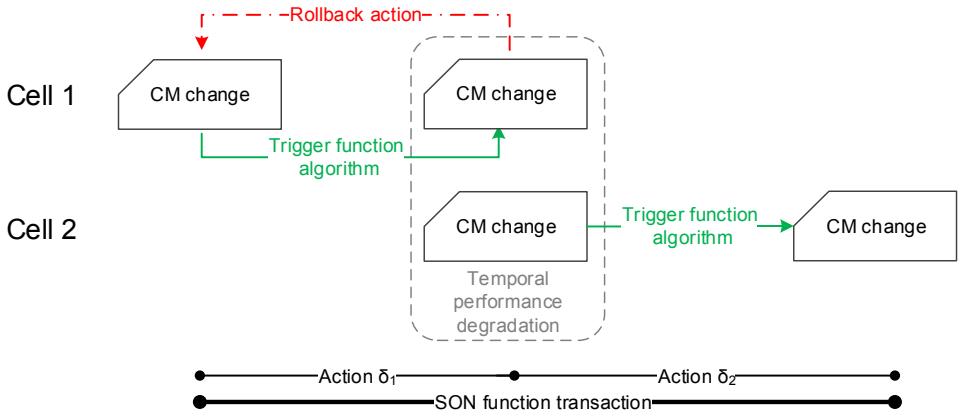


Figure 3.3: Example of a SON function transaction

actually not applied. An example is the MRO function which also relies on a proper coverage setup.

As a result, a verification strategy must be able to identify such transactions in order to prevent interferences with ongoing optimization processes.

3.2.2 Verification Collisions

CM changes have an impact not only on the reconfigured cell, but also on those surrounding it. For instance, in the concept of SON coordination (cf. Section 2.3.2.3) the cells that have been influenced by a SON function's action, are added to the impact area of the given function change. Note that the term impact area originates from the SON coordination concept and describes which parts of the network are affected by the change. A popular example is the CCO function whose action may affect all direct neighbors of the cell that has been reconfigured. This means that concurrent conflicting actions should not only be prevented on the cell whose configuration has been adjusted, but also on those surrounding it. Figure 3.4 visualizes this statement. The two actions $\delta_i \in \Delta_i, \delta_j \in \Delta_j$ must not be simultaneously deployed since they are marked as conflicting and have a potential impact on each other. On the contrary, $\delta_k \in \Delta_k$ is permitted.

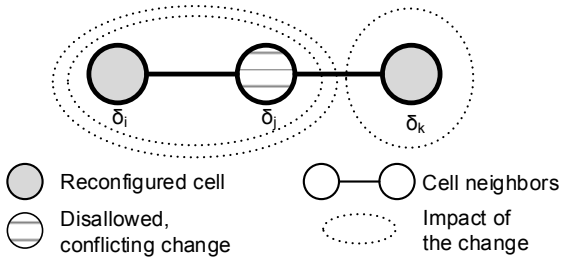


Figure 3.4: Conflict example before applying changes

However, rolling back already deployed changes is more than re-applying a set of configuration snapshots. First of all, there has to be a reason why we are undoing changes made in the past. The cause is typically an anomaly, e.g., a degradation in performance, that a cell or a set of cells start to experience after the execution of certain changes. Second, we need to know which actions we need to rollback, i.e., we need to analyze the detected problem and find a corrective action. However, during the process of finding the appropriate rollback actions, we may encounter *uncertainties* as given by following example. Let us assume that we have 10 cells, as given in Figure 3.5. Three of those cells (1, 2, and 3) have been reconfigured whereas another three (5, 8, and 9) have degraded. In addition, let us assume that cell 2 is not responsible for any degradation, which we do not know. As a result, we have three potential rollbacks actions, one being generated for each reconfigured cell.

A simplistic solution is rolling back all changes. However, by doing so we may also revert changes that were necessary and did not harm performance. As a result, we have

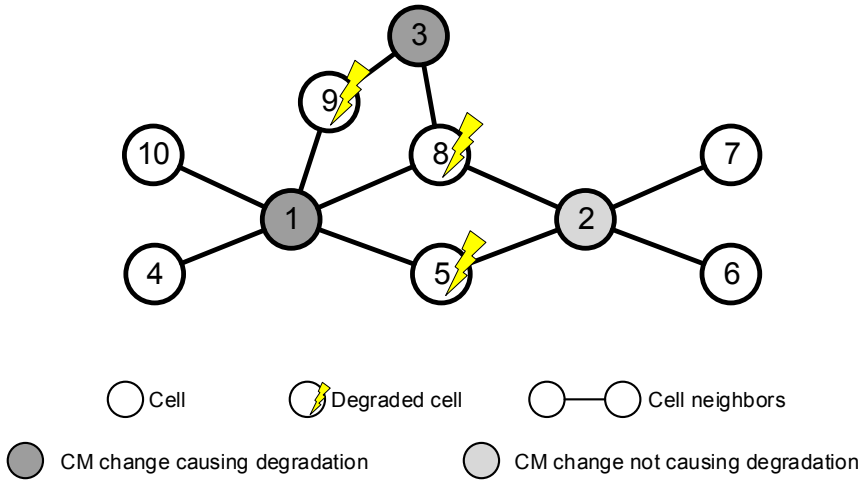


Figure 3.5: Example of a verification collision

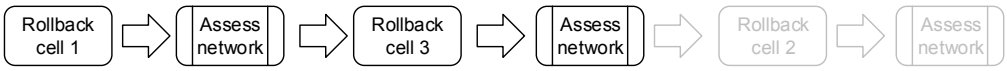


Figure 3.6: Example of a sequence of corrective rollback actions. The actions colored in black are executed whereas those in gray are unnecessary and dismissed.

an uncertainty which rollback action to execute, which to possibly delay, or even omit. This uncertainty is also being referred to as a *verification collision* which can be further specified as follows:

Definition 3.3 (Verification collision)**.** Let us denote the set of all cells as Σ and the set of all corrective rollback actions as $C^{\perp}$. Furthermore, let ζ be a function that returns the initial trigger for the rollback action, i.e., the cells that led to the generation of a rollback action $c^{\perp} \in C^{\perp}$, as follows:

$$\zeta : C^{\perp} \rightarrow \mathbb{P}(\Sigma) \setminus \emptyset, \tag{3.1}$$

where $\mathbb{P}(\Sigma)$ is the power set of the cell set Σ. Two rollback actions $c_i^{\perp}, c_j^{\perp} \in C^{\perp}$ are said to be in collision if and only if $\zeta(c_i^{\perp}) \cap \zeta(c_j^{\perp}) \neq \emptyset$.

In order to resolve such collisions, we need to *serialize* the process of executing corrective rollback actions. If we take the simplified scenario from above, a permissible sequence of corrective actions would be to deploy a rollback for cell 1, and 3, as well as dismiss the one generated for cell 2. Figure 3.6 visualizes this sequence.

To define such a sequence, we need to split the set $C^{\perp}$ by assigning its elements to disjoint and non-empty subsets, i.e., the set of all rollback actions $C^{\perp}$ has to be *partitioned* [Bru09]. The result is called a *corrective action plan*.

Definition 3.4 (Corrective action plan)**.** A corrective action plan for the set of rollback actions $C^{\perp}$ is a grouping of the set's elements into non-empty subsets $\mathcal{P}^{C^{\perp}} = \{\hat{C}_1^{\perp}, \ldots, \hat{C}_i^{\perp}\}$, for which the following properties must hold:

- $\emptyset \notin \mathcal{P}^{C^{\perp}}$
- $\bigcup_{\hat{C}^{\perp} \in \mathcal{P}^{C^{\perp}}} \hat{C}^{\perp} = C^{\perp}$
- if $\hat{C}_1^{\perp}, \hat{C}_2^{\perp} \in \mathcal{P}^{C^{\perp}}$ and $\hat{C}_1^{\perp} \neq \hat{C}_2^{\perp}$ then $\hat{C}_1^{\perp} \cap \hat{C}_2^{\perp} = \emptyset$

The number of different plans, i.e., partitions of the set $C^{\perp}$, is given the Bell number B_n, where n is the number of elements of the set [FS09]. For instance, B_7 gives us 877 possible combinations. In addition, the actions assigned to the same subset must not be in collision with each other, which is called a *collision-free corrective action plan*.

Definition 3.5 (Collision-free corrective action plan)**.** The collision-free property of a corrective action plan $\mathcal{P}^{C^{\perp}} = \{\hat{C}_1^{\perp}, \ldots, \hat{C}_i^{\perp}\}$ is expressed as follows:

- $\forall c_i^{\perp}, c_j^{\perp} \in \hat{C}^{\perp} \colon \zeta(c_i^{\perp}) \cap \zeta(c_j^{\perp}) = \emptyset$

If we recall the afore-mentioned example, we would have three subsets, each containing exactly one element, i.e., $\forall i \colon |\hat{C}_i^{\perp}| = 1$. In this thesis, such a plan is referred to as a *completely serialized corrective action plan*. Furthermore, the plan must have the property of being *gain-aware*.

Definition 3.6 (Gain-aware corrective action plan)**.** The gain-aware property of a corrective action plan $\mathcal{P}^{C^{\perp}} = \{\hat{C}_1^{\perp}, \ldots, \hat{C}_i^{\perp}\}$ is expressed as follows:

- $\mathcal{G}(\hat{C}_1^{\perp}) \geq \cdots \geq \mathcal{G}(\hat{C}_i^{\perp})$, where $\mathcal{G}$ returns the gain of deploying the set of actions $\hat{C}_i^{\perp}$.

3.2.3 Over-Constrained Corrective Action Plan

In the previous section, the verification collision term was introduced and the definition of a corrective action plan was provided. The properties of being completely serialized and collision-free have been discussed as well. Let us continue with the latter one. The collision-free property of a corrective action plan $\mathcal{P}^{C^{\perp}} = \{\hat{C}_1^{\perp}, \ldots, \hat{C}_i^{\perp}\}$ can be expressed as a set of constraints that must not be violated. In particular, every verification collision is a constraint that prevents two corrective actions from being simultaneously deployed. As we saw earlier, the more constraints we have, the more sets $\{\hat{C}_1^{\perp}, \ldots, \hat{C}_i^{\perp}\}$ we get, i.e., the steps required to process the plan increase. This can reach up to a point where we have to sequentially process each corrective action, as presented in Figure 3.6. The plan contains three rollback actions and $|\mathcal{P}^{C^{\perp}}| = 3$, i.e., $\mathcal{P}^{C^{\perp}}$ is completely serialized.

Unfortunately, we also have a constraint on the maximum number of sets $\{\hat{C}_1^{\perp}, \ldots, \hat{C}_i^{\perp}\}$, i.e., an upper limit for $|\mathcal{P}^{C^{\perp}}|$. It may originate from the time available for deploying

rollback actions, for instance, we may be allowed to restore settings only once a day. Another could be the environment, e.g., in a highly populated area we may have the requirement of restoring the network performance as fast as possible. As a result, we may not be able to process all actions, as depicted in Figure 3.7. It shows the impact on the action plan if we add the constraint $|\mathcal{P}^{C^\perp}| < 2$. Such a plan is also referred to as an *over-constrained corrective action plan*.

Definition 3.7 (Over-constrained corrective action plan)**.** A corrective action plan $\mathcal{P}^{C^\perp} = \{\hat{C}_1^\perp, \ldots, \hat{C}_i^\perp\}$ is called over-constrained if and only if its size exceeds the maximum number of allowed sets of corrective rollback actions $\hat{C}^\perp$. If we denote this limit as τ, $|\mathcal{P}^{C^\perp}| > \tau$ must hold for an over-constrained corrective action plan.

Having such an action plan means that we cannot find a partition of $\mathcal{P}^{C^\perp}$ that satisfies all given constraints, i.e., we have an *over-constrained verification problem*. In order to find a solution we would need to relax the given constraints, i.e., to find a good compromise that *minimizes the total constraint violation*. In the terms of a verification process, we would need to identify *soft verification collisions*.

Definition 3.8 (Soft verification collision)**.** A soft verification collision $C^\perp \times C^\perp$ is a valid constraint, that prevents two actions $c_i^\perp, c_j^\perp \in C^\perp$ from being simultaneously executed, but whose removal is necessary for minimizing the total constraint violation of putting $|\mathcal{P}^{C^\perp}|$ in the range of $[1; \tau]$.

However, violating constraints adds a new set of issues that we would need to consider. The question that arises is which criteria to use for minimizing the impact of violating rollback conflicts. Furthermore, removing a constraint means that we will permit the simultaneous execution of rollback actions which was initially not allowed to happen. As a consequence, the risk of rolling back changes that did not do any harm to the network increases. For instance, it is possible to get an execution order as the one presented in Figure 3.8. It depicts a permissible outcome after we remove the verification collision between the rollbacks for cell 1 and 2 (cf. Figure 3.5). Instead of omitting the rollback for cell 2, it is placed at the very beginning of the corrective action sequence.

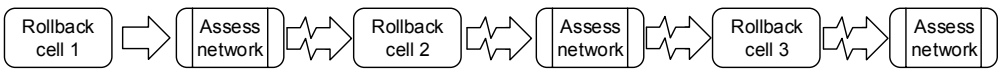


Figure 3.7: Adding constraints to a corrective action plan may result in the inability of processing all rollback actions. Due to the constraint $|\mathcal{P}^{C^\perp}| < 2$, it is impossible to process all actions.

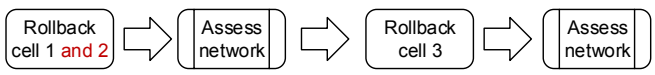


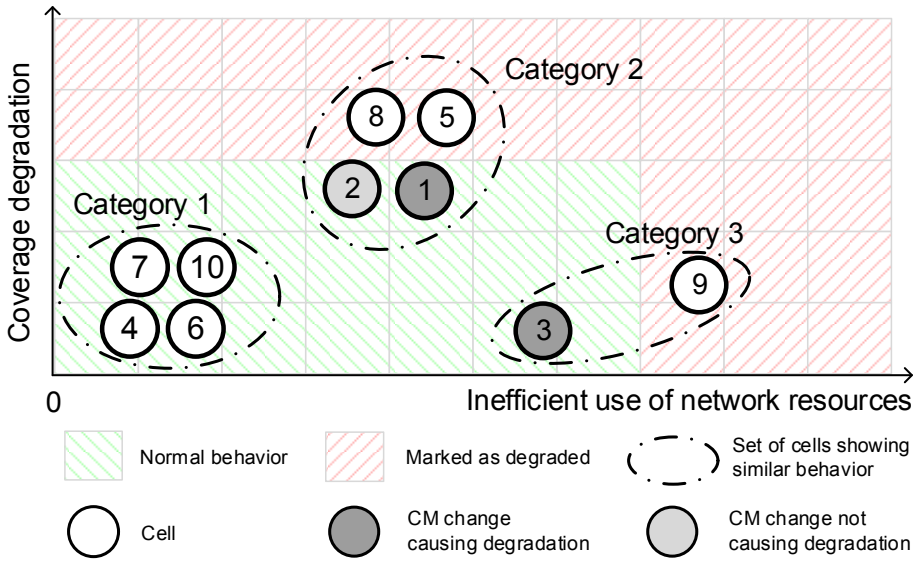
Figure 3.8: The impact of verification collision removal. The elimination of the collision between the corrective actions for cell 1 and 2 may lead to the execution of both actions at the same time.

3.2.4 Weak Verification Collisions

Let us continue with the problem of having an over-constrained corrective action plan (cf. Definition 3.7). Being unable to process all elements of $\mathcal{P}^{C^{\perp}} = \{\hat{C}_1^{\perp}, \ldots, \hat{C}_i^{\perp}\}$, because $i > \tau$, does not necessarily mean that we have lowered τ too much. Instead, we may have been too strict when identifying verification collisions. Let us give an example by observing the performance statistics of the cells from Figure 3.5. Let us assume that they include the coverage degradation reports of the cells as well as how cells utilize network resources. A prominent example for inefficient use of network resources are unnecessary handovers, also called ping-pongs. They are repeated between two cells within a short time, leading to reduced user throughput [NTB+07].

However, if we try to group cells based on such performance indicators, we may get an outcome as presented in Figure 3.9(a). It shows an exemplary position of each cell in the $\mathbb{R}^2$ space. For this particular example, we can group the cells into three categories:

- Category 1: Cells showing normal behavior.
- Category 2: Cells experiencing coverage degradation.
- Category 3: Cells inefficiently using network resources.



(a) Cell performance visualized in the $\mathbb{R}^2$ space. The collision between cell 1 and 3 may be removed due to the grouping of the cells.

(b) The resulting sequence of corrective actions after the elimination of the collisions. The configurations of cell 1 and 3 are rolled back whereas the changes at cell 2 are accepted.

Figure 3.9: Correlation between cell performance, verification collisions, and corrective actions

As shown, cells 1, 2, 5, and 8 are clearly experiencing coverage degradation whereas cell 3 and 9 an inefficient use of resources. Hence, neither cell 1 has been assigned to the group of cell 3 and 9, nor has cell 3 been grouped together with cell 2 and 8. In other words, we may consider removing the verification collision between cell 1 and 3 as well as between 2 and 3, which will give us a sequence of corrective actions as shown in Figure 3.9(b).

Obviously, those collisions are unneeded and unnecessarily delay the process of executing the corrective actions and, therefore, the restoration of the network performance. In this thesis, they are referred to as *weak verification collisions*[1].

Definition 3.9 (Weak verification collision)**.** A weak verification collision $C^{\perp} \times C^{\perp}$ is a constraint that prevents two actions $c_i^{\perp}, c_j^{\perp} \in C^{\perp}$ from being simultaneously executed, but whose removal may not lead to the rollback of changes not harming performance. That is, there is the possibility for such a collision to be a false positive one.

3.2.5 Dynamic Topology Changes

Until now, the major topic of discussion was the assessment of configuration changes and the provision of a set of corrective *rollback* actions in case they harm performance. However, reconfigurations that lead to topology changes have been neglected so far. Such changes may have an even more serious impact on a verification process compared to those discussed in the previous sections.

Today, one of the most common reasons why we have dynamic topology changes in a mobile network is energy saving [3GP10, MSES12]. Cells are activated or deactivated based on the current demands of the network, for instance, when a large group of UEs suddenly leaves the network some small cells can be turned off. However, disabling or enabling cells creates uncertainties during the process of detecting anomalies as well as assembling and executing a corrective action plan. Let us start with the example depicted in Figure 3.10. It represents three cells, each being a neighbor of the other two. In addition, cell 2 has been disabled and cells 1 and 3 are showing an anomalous behavior.

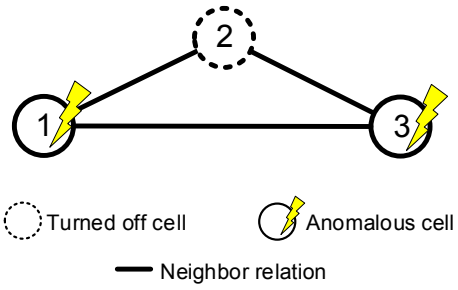


Figure 3.10: The impact of topology changes on cell performance. Turning off a cell may induce anomalies in a similar way as CM changes do.

[1] Weak collisions are also referred to as false positive verification collisions [TATSC16a].

Obviously, if we do not have any other events in the network, we would assume that switching off cell 2 is the cause for that behavior. This assumption could be motivated by the state of the available cell profiles. The profile itself is defined in the following way:

Definition 3.10 (Profile)**.** The profile of a cell is a vector $\mathbf{p} = (p_1, \ldots, p_i)$, where p_i specifies the expected value or range of the i^{th} cell KPI.

Generally speaking, the profile defines how cells should usually behave. Furthermore, it specifies how a cell should behave with respect to its neighbors, as well as determines the expected performance from those neighbors. It should be noted that the assessment of the neighbor performance usually requires the analysis of mobility related KPIs like the handover ping-pong rate or the handover success rate [3GP16f, HSS11].

To record a profile, we need a training phase during which KPI data is collected. The gathered data is also referred to as a *profile input*.

Definition 3.11 (Profile input)**.** The input of a profile is defined as a matrix $\mathbf{K}_{i,j}$ (Equation 3.2) in which the rows i correspond to the KPIs that are used for profiling, whereas the columns j represent discrete points in time when KPIs data was exported. The total number of columns depends on the PM granularity period and the profile record duration.

$$\mathbf{K}_{i,j} = \begin{pmatrix} k_{1,1} & k_{1,2} & \cdots & k_{1,j} \\ k_{2,1} & k_{2,2} & \cdots & k_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ k_{i,1} & k_{i,2} & \cdots & k_{i,j} \end{pmatrix} \tag{3.2}$$

However, if we add or remove a neighbor of a cell, the current assumptions about the network and the one made while recording a profile may no longer match. As a result, we get an *incomplete profile*.

Definition 3.12 (Incomplete profile)**.** A cell profile is called incomplete if and only if $|\Sigma|_{\mathbf{K}} \neq |\Sigma|_l$, where Σ is the set of all cells, $|\Sigma|_l$ the current number of cells, and $|\Sigma|_{\mathbf{K}}$ the number of cells that were present when the profile input was collected.

The most logical solution would be to update the current profile or specify a new one. However, if cell 2 needs to be turned on again, we will face the same problem. In other words, we have the question which cell state is actually the expected one. If we assume that the initial state, i.e., when cell 2 was enabled, is the usual one, we would need to rollback the turn off action. However, this could be an inappropriate action since cell 2 is no longer required in the network and has been, therefore, put into energy conserving mode.

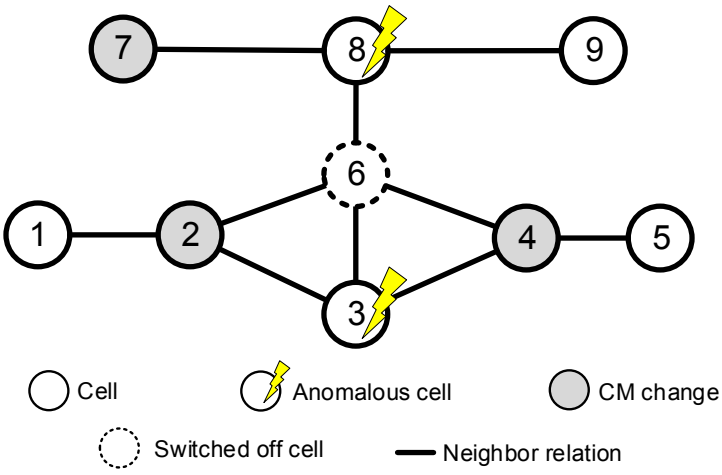


Figure 3.11: Visualization of concurrent topology and CM changes. They lead to uncertainties about the cause of the anomalous behavior of cells.

Theoretically, we could define several profiles, e.g., for every possible topology change. In the aforementioned example this would be a rather straight forward procedure. However, this changes for a real mobile network since it contains hundreds of cells [BKKS16, NTSM15] which may also have a high neighborship degree. Consequently, updating incomplete profiles would be a highly complex task since we would need to do that for numerous cell combinations. In particular, if all elements of the set of all cells Σ are allowed to change their availability, we would require $|\mathbb{P}(\Sigma)| - 1$ profile combinations. If we denote $|\Sigma| = m$ the resulting value would be $2^m - 1$. Note the power set includes the empty set $\emptyset$ which is why $|\mathbb{P}(\Sigma)|$ is decreased by 1.

Unfortunately, the fact that we can have incomplete profiles is not the only challenge that we would need to overcome. Let us assume that we have a network consisting of 9 cells and 10 cell adjacencies, as shown in Figure 3.11. Moreover, suppose that cells 2, 4, and 7 have been reconfigured, cell 6 is disabled, and cells 3 and 8 have degraded. Hence, for this particular network snapshot we have three rollback actions $c_2^\perp, c_4^\perp, c_7^\perp$ and a verification collision between $c_2^\perp$ and $c_4^\perp$. Note that index i of an action $c_i^\perp$ represents the ID of the cell for which it has been generated. If we assume that network configuration changes have caused the anomalies, a permissible corrective action plan $\mathcal{P}^{C^\perp}$ would be $\{\{c_2^\perp, c_7^\perp\}, \{c_4^\perp\}\}$ or $\{\{c_4^\perp, c_7^\perp\}, \{c_2^\perp\}\}$.

However, those assumptions may not be always valid. Besides CM changes there is a certain event type that may induce an anomaly and which we even cannot correct by executing a rollback action. This unusual event is caused by UE movements, that is, users frequently entering or leaving the network. If a rather large group of UE joins or leaves, it may induce anomalous cell behavior, like an unusually high or low load and throughput. Hence, the anomalies at cell 3 and 8 might be caused by one or more UE groups that have recently joined or left the network.

As a result, the above-mentioned plan $\mathcal{P}^{C^{\perp}}$ cannot restore the network performance even if we execute all suggested changes. In this thesis, this plan is called a *weak corrective action plan*. It can be defined as follows:

Definition 3.13 (Weak corrective action plan)**.** A corrective action plan $\mathcal{P}^{C^{\perp}} = \{\hat{C}_1^{\perp}, \ldots, \hat{C}_i^{\perp}\}$ is called weak when the set of anomalous cells remains nonempty, i.e., $\Sigma^A \neq \emptyset$, after processing $\forall \hat{C}_i^{\perp} \in \mathcal{P}^{C^{\perp}}$.

3.3 Factors Influencing a Verification Process

In this Section, the factors that influence a verification process and hinder its proper operation are discussed. Concretely, the major topic of discussion includes the availability of PM and CM data and the statistical relevance of PM data. They are introduced in Sections 3.3.1 and 3.3.2, respectively.

Moreover, the heterogeneous nature of today's networks and the characteristics of CM parameters play an important role during verification. Sections 3.3.3 and 3.3.4 are devoted to those topics.

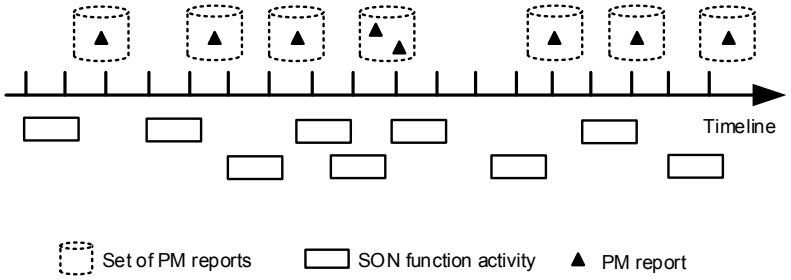
3.3.1 Availability of PM and CM Data

3.3.1.1 PM Granularity Periods

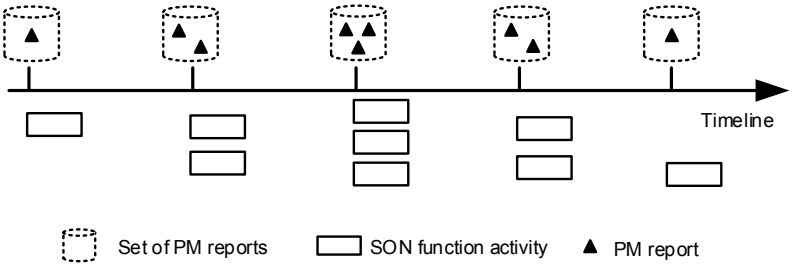
A factor that influences the ability to assess the impact of CM changes is the time that is required to get PM data from the network, which is also known as a PM granularity period (cf. Section 2.2.3). The more frequently PM data is exported from the network, the more often SON functions can trigger their monitoring part, observe the KPIs of interest, and make CM changes. Figure 3.12(a) shows an exemplary function activity in such a case. As we can see, functions may not be required to run in parallel which minimizes the necessity of analyzing numerous CM changes at the same time.

However, if the granularity period is increased we may face a function activity like the one depicted in Figure 3.12(b). A long granularity period increases the probability of SON functions to become active at the same time. Hence, numerous configuration changes can be simultaneously deployed which increases the likelihood of verification collisions to emerge. Also, the issues that occur during the process of generating a corrective action plan remain, e.g., an over-constrained corrective action plan or the presence of weak collisions.

Romeikat et al. [RSB13] estimated that the lower bound of getting PM data in a OAM system is 15 minutes which is a rather long time frame. The main reason for that is the relative network and processing overhead of gathering the PM data and uploading it to the corresponding database. It should be noted that the OAM architecture is presented in Section 2.2.1.



(a) Short granularity periods, i.e., frequent access to PM data, may minimize the likelihood of SON to run parallel.



(b) Long granularity periods, i.e., rare access to PM data, may lead to simultaneous CM changes which may increase the likelihood of verification collisions to emerge.

Figure 3.12: Correlation between granularity periods, verification collisions, and the activities of SON functions

3.3.1.2 Access to CM Data

In order to observe as much as possible ongoing CM changes, a verification process needs to have a wide view of the network. In terms of the 3GPP OAM architecture, as introduced in Section 2.2.1, it would mean that it needs to monitor the NEs from the DM or even the NM level.

However, being at that level induces delays until it gets relevant data from the network. In particular, it would not be able to instantly observe the impact of CM changes, but will have to assess a set of configuration actions that have been collected over time. Figure 3.13 illustrates the resulting view on CM and PM data. Instead of assessing each of the nine changes immediately after they have been executed, it will do that only twice: for five and four changes, respectively.

Unfortunately, the analysis of a large set of configuration changes at once increases the likelihood for verification collisions to emerge. Hence, the problems of having an over-constrained corrective action plan as well as weak verification collisions remain. Concretely, we are facing the same issues as discussed in Section 3.3.1.1.

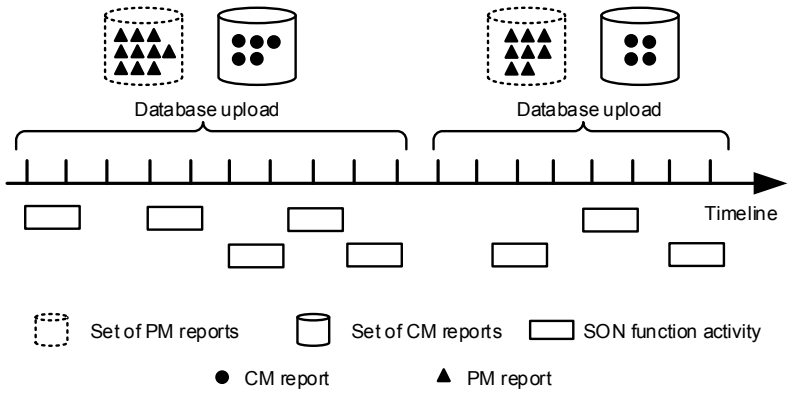


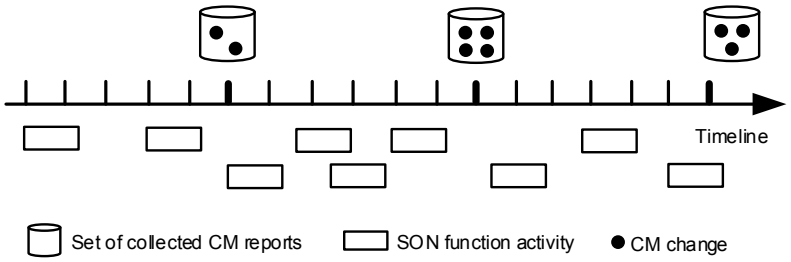
Figure 3.13: The consequences of having a coarse granular view on CM data. PM data has to be collected until access to CM data is provided. Also, numerous CM changes are assessed by a verification process at the same time.

3.3.2 Statistical Relevance of PM Data

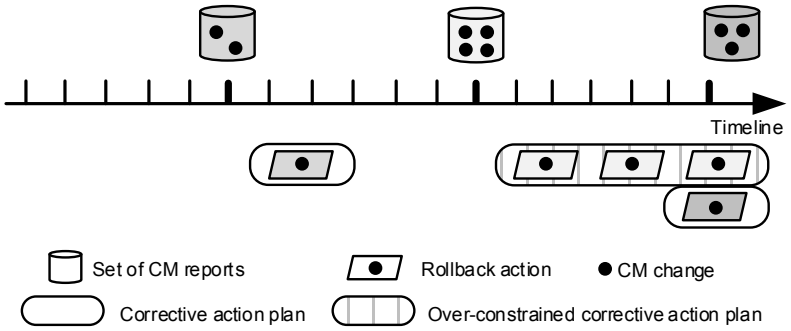
One of the prerequisites for being able to assess the impact of configuration changes is the statistical relevance of PM data. Typically, we would pick several KPIs and monitor their behavior after the reconfiguration has been carried out. Nevertheless, significant changes in certain KPIs cannot be immediately seen. For instance, a significant change in KPIs like the handover success rate or the handover ping-pong rate can be only observed when enough UEs actively use the network, e.g., during peak hours of a weekday.

Therefore, we would need to accumulate the changes that have been made over a longer time frame and assess them at once after we are in possession of such PM data. A visualization of this process is shown in Figure 3.14(a). Instead of assessing CM changes after each SON function activity, we would do so only three times. Af first, we would observe two, then four, and finally three changes. As stated before, this would also increase the chances of getting verification collisions (cf. Definition 3.3).

Furthermore, this may also result in an over-constrained corrective action plan (cf. Definition 3.7). Suppose that some of the actions need to be rolled back, as shown in Figure 3.14(b). Each set of rollback actions can be seen as a trigger for creating a corrective action plan $\mathcal{P}^{C^{\perp}} = \{\hat{C}_1^{\perp}, \ldots, \hat{C}_i^{\perp}\}$. In the presented example, the size of the first and third one would be one, i.e., it contains only one set of rollback actions $\hat{C}_i^{\perp}$. The size of the second one is three which unfortunately overlaps with the third corrective action plan. As a result, we would need to reduce the size of the second one, i.e., we get an over-constrained verification collision problem. It should be noted that this issue is presented in Section 3.2.3.



(a) The absence of statistically relevant PM data leads to the accumulation of CM changes. All of the accumulated changes must be assessed by a verification process.



(b) The accumulation of CM changes may lead to an over-constrained corrective action plan. As a result, it cannot be processed for the given time.

Figure 3.14: The impact of absent statistically relevant PM data on a verification process

3.3.3 Network Density, Heterogeneity, and Topology

Nowadays, mobile networks are comprised of a high number of cells as well as a high number of cell adjacencies. For instance, in [BKKS16] datasets from a real LTE mobile network have been acquired. They contain performance, configuration, fault, and alarm measurements for approximately 4000 cells. In [TSC15], a similarly sized LTE network has been observed. It consists of 3028 cells as well as a high number of neighbor relations, as the cell out-degree distribution in Figure 3.15 depicts. Another example is given in [CCC+14a], where the observed 3G network consists of roughly 2000 cells.

The reasons for having a high number of cells and cell adjacencies are numerous. Many of today's networks are heterogeneous networks. They are characterized by a mixture of different RATs, like the GSM, HSPA, and LTE (cf. Section 2.1.1). Hence, multi-RAT neighbor relations have to be established in order to hand over UEs between networks of different RATs. Furthermore, heterogeneous networks are multi-layer, i.e., in addition to macro cells, we have numerous micro and femto cells that provide hot-spot coverage. Those cells can be placed in areas of dense traffic, for example, to cover peak hour traffic demands. Moreover, future mobile networks may include even more heterogeneous networks, different types of access technologies, as well as a high number of small cells densely clustered together [Nex15, CZ15].

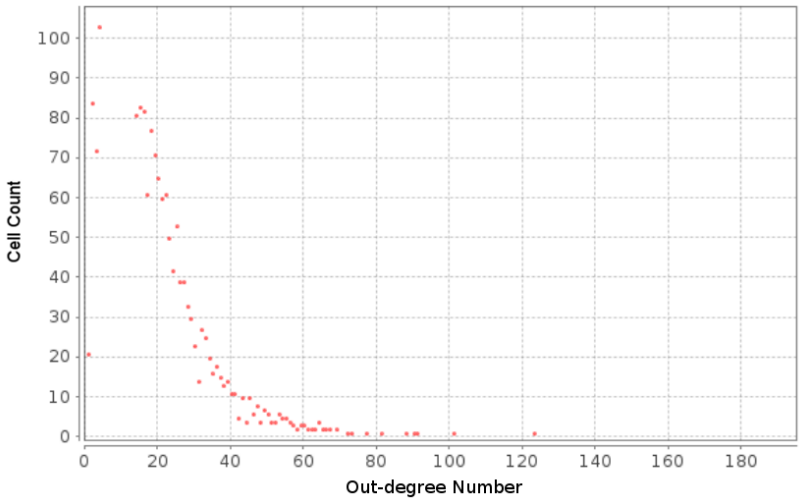
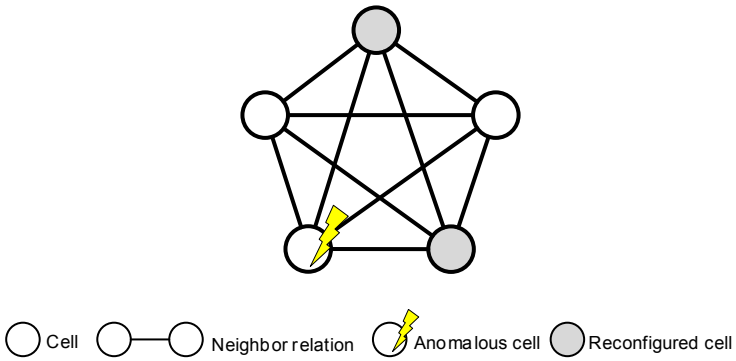


Figure 3.15: Cell out-degree distribution of an LTE network [TSC15]. On the x-axis the cell out-degree bins are shown whereas on the y-axis the number of cells that fall within each bin.

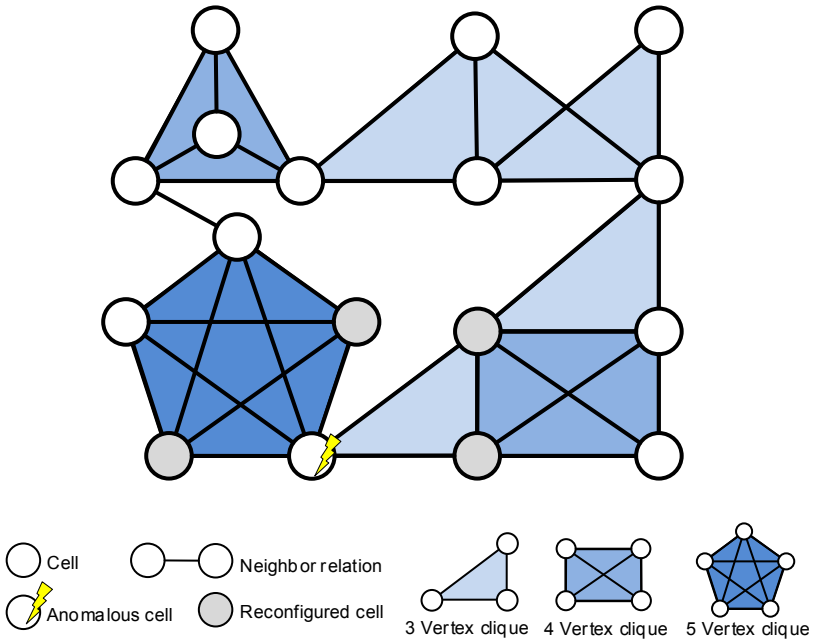
However, having such networks also complicates the process of discovering suboptimal network performance. It becomes a highly complex task due to the fact that real deployments are actually an overlay of different RATs. According to Bosneag et al. [BHO16], the relatively large size of today's mobile networks requires not only the monitoring of many important performances counters, but also the consideration of different cell configurations settings and environmental changes. It comes mainly from the numerous scenarios and parameter combinations that can emerge and which have to be taken into consideration.

Highly dense networks also complicate the process of diagnosing why cells are anomalously performing, i.e., the generation of a corrective action. Let us consider the example from Figure 3.16(a). It represents a graph in which any two cells have a neighbor relation between each other. Hence, we have a complete graph [CLRS09] in which every pair of distinct vertexes is connected by an edge, i.e., the density is 1.0 since the graph has the maximum number of edges. Consequently, such a network topology always leads to a verification collision if at least two cells are reconfigured and one has become anomalous, as shown in the figure.

Nevertheless, even if we have a low graph density we can still face the same problem. Suppose that we have a topology like the one presented in Figure 3.16(b). Despite the fact that the graph has a density of 0.27, it consists of several areas in which every two cells have a common neighbor relation. In terms of graph theory, such areas can be referred to as cliques [CLRS09], i.e., a subset of vertexes in an undirected graph that induces a complete subgraph. In other words, a verification process will face the same issues as described before.



(a) Verification collisions in a complete graph



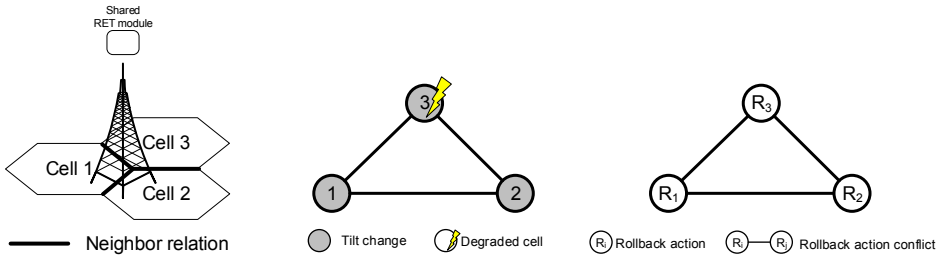
(b) Verification collisions in a graph of multiple cliques

Figure 3.16: Graph analysis of the verification collision problem

3.3.4 Characteristics of CM Parameter Changes

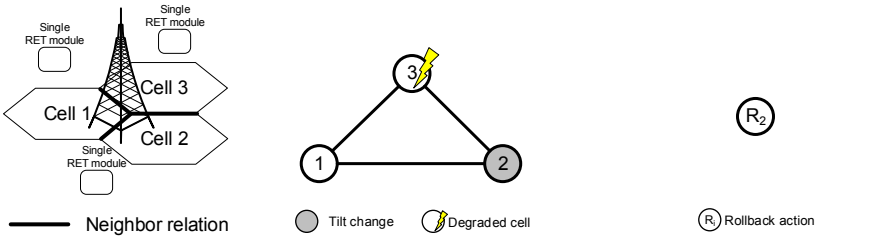
3.3.4.1 CM Change Granularity

In [NTSM15], a performance test of an offline CCO algorithm is made. The authors observed the network for anomalies after the deployment of the suggested up or down tilt changes. In total, 21 WCDMA cells were selected by the algorithm for reconfiguration. Furthermore, the tilt changes were deployed by using shared Remote Electrical Tilt (RET) modules, i.e., modules to which more than one antenna is connected. Having a shared module, though, prevents the separate change of the tilt angels of the antennas. As a consequence, the CCO algorithm had to assemble a new configuration for all cells



(a) Network overview (b) Network activities (c) Rollback action conflicts

Figure 3.17: Example of having one shared RET module



(a) Network overview (b) Network activities (c) Rollback action conflicts

Figure 3.18: Example of having three single RET modules

managed by the same RET module which resulted in 32, instead of the initially suggested 21 changes.

Although the authors did not find an anomaly after the deployment of the CCO changes, care should be taken when in the presence of shared modules changes are rolled back. Suppose that three cells are managed by a single module, as shown in Figure 3.17(a), and that a tilt change is required for only one of the cells. Due to the shared module we will have three tilt changes, as depicted in Figure 3.17(b). Should one of the cells degrade, we will get three rollback actions, each being in collision with the other, as outlined in Figure 3.17(c).

On the contrary, if each cell is managed by a separate single module, i.e., we have fine granular configuration options, as given in Figure 3.18(a), we would lower the probability of getting collisions. As shown in Figures 3.18(b) and 3.18(c), we have only one rollback action, i.e., no collisions at all.

3.3.4.2 Impact of CM Changes

Let us start with a simplistic example where we have two active SON functions in the network: MRO and CCO. Note that a description of those functions is given in Section 8.1.2. If the CCO function modifies the antenna tilt, the cell border changes physically which means that the received signal quality changes as well. Obviously, this affects the handover capabilities of the neighbors of the reconfigured cell which

could be monitored by the MRO function. As a result, any upcoming CIO changes suggested by that function are influenced by tilt adjustments made in the past. In addition, such function changes have to be properly coordinated since otherwise they may cause configuration, measurement or logical conflicts (cf. Section 2.3.2.3). Omitting that can potentially cause undesired network behavior [RSB13], e.g., the above-mentioned MRO function should run after CCO has made its changes.

Similarly, a corrective rollback action is a CM change that may trigger inactive SON functions as well as affect their outcome. For instance, reverting an antenna tilt configuration is a CM change that can trigger the MRO function monitoring the same physical area. As a consequence, we have an indirect constraint on a corrective action plan $\mathcal{P}^{C^\perp} = \{\hat{C}_1^\perp, \ldots, \hat{C}_i^\perp\}$ that forces us to deploy the correct rollbacks already after processing the first rollback action sets $\hat{C}_i^\perp$. Otherwise, we can trigger functions for which the environment is not prepared yet.

Of course, it is possible to block SON functions in the area where a verification process is running, e.g., by using a SON coordinator [Ban13]. However, the question that arises here is whether this is applicable in a real network. If we consider the size of such networks, we can quite delay necessary optimization changes for a high number of cells, as depicted in Figure 3.19. The rollbacks for the degraded cells block any other SON function activity within the same area.

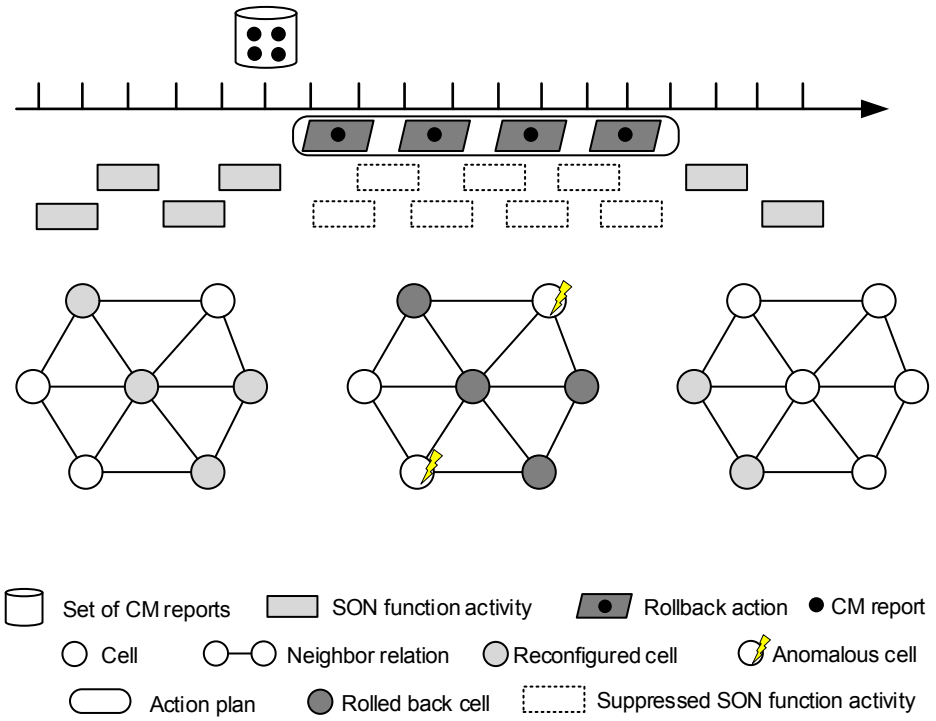


Figure 3.19: Example of blocking a SON optimization process when processing a corrective action plan

3.4 Summary

In today's mobile networks, there are several sources that can lead to abnormal cell performance (cf. Section 3.1). The issues that are most difficult to troubleshoot are configuration changes that have been deployed to the network while it was operating, and harmed its performance. The reasons why there is a possibility to deploy such changes are manifold. On the one hand, offline SON algorithms are unable to generate optimal CM settings in case they are supplied with inaccurate knowledge about the network. On the other hand, online SON techniques have a limited view on the ongoing network changes and are optimizing the network only locally.

In this chapter, the major topic of discussion was the detailed analysis of the problems that emerge while assessing the impact of already deployed configuration changes on the network performance, as well as rolling back those harming it. A strategy like that is referred to as a verification process (cf. Definition 3.1) whose purpose is to observe CM changes, monitor KPIs, and generate a corrective action plan in case cells start showing an anomalous behavior, like a degradation in performance. The corrective action itself is a rollback to a previous stable configuration state.

Many of today's SON troubleshooting and anomaly detection & diagnosis approaches already follow the basic principles of a verification process. However, they experience issues while providing a corrective action plan, as the following answer outlines:

O1.1: Investigate and study the disadvantages and weaknesses of approaches that verify network operations.
Q: What are disadvantages of today's anomaly detection & diagnosis and SON troubleshooting approaches?
A: SON troubleshooting and anomaly detection & diagnosis approaches that are being used today cannot provide one single action since they need to know in advance the combined performance impact of the rolled back CM changes. To do so, they would not only require a sophisticated diagnosis component, but also *reusable* diagnosis results. Getting such results, however, is a complex task. The reason is the nature of today's mobile networks: they are manifold, generate different types of PM data and react differently to the same corrective action. Hence, such methods have to deploy corrective action through several iterations, i.e., they create a sequence of corrective actions. Having a sequence of such actions, however, means that another SON optimization process may interfere in the meantime and prevent the necessary corrective actions from being deployed. Furthermore, uncertainties may occur while processing the corrective actions.

The uncertainties that are mentioned above are referred to as verification collisions which are addressed by the following research questions:

O3.1: Define uncertainties in the terms of a verification process.
Q: What is a verification collision?
A: A verification collision (cf. Definition 3.3) is an uncertainty which corrective rollback action to execute. It occurs when two rollback actions share the same trigger, i.e., cells that led to the generation of the two actions.

Q: Why do verification collisions emerge?
A: The reasons why verification collisions emerge are manifold. First, the access to PM and CM data plays an important role (cf. Section 3.3.1). A coarse-granular view on performance and configuration data increases the likelihood of verification collisions to appear since multiple CM changes need to be simultaneously assessed by a verification process. Second, the availability of statistically relevant PM data is also a factor that leads to the formation of collisions (cf. Section 3.3.2). A significant change in KPIs is only visible when the network is monitored for a longer period of time. As a result, ongoing CM changes cannot be immediately assessed, but have to be collected and simultaneously verified when such data is available. Third, the heterogeneous nature of today's mobile networks (cf. Section 3.3.3), in particular, the resulting cell interconnectivity increases the probability of getting verification collisions. Fourth, the way of how CM changes are deployed today may also cause verification collisions to emerge (cf. Section 3.3.4). In particular, due to shared reconfiguration modules one change may actually result into multiple CM adjustments.

Unfortunately, the presence of verification collisions has a negative impact on the resulting corrective action plan, as outlined below.

O3.2: Study and evaluate the impact of uncertainties on the outcome of a verification process.
Q: How do verification collisions impact the corrective action plan?
A: A corrective action plan (cf. Definition 3.4) must have the property of being collision-free (cf. Definition 3.5). However, the presence of numerous verification collisions increases the number of sequentially processed corrective actions which may lead to an over-constrained plan (cf. Definition 3.7). Hence, it is frequently impossible to solve the verification problem without neglecting some verification collisions, i.e., marking some collisions as soft ones (cf. Definition 3.8).

Q: What are the consequences of neglecting verification collisions?
A: Neglecting verification collisions may result in the rollback of configuration changes that were necessary and did not harm performance. Furthermore, it may result in suboptimal changes in the future (cf. Section 3.2.1).

Q: Are all verification collisions justified?
A: No. There are also weak collisions (cf. Definition 3.9), which can be potentially false positives, i.e., they may unnecessarily delay a verification process.

However, verification collisions are not the only type of uncertainties that may emerge during the process of verification. Dynamic changes in the network topology create a very similar problem, as discussed below.

O4.1: Specify and define the uncertainties caused by topology changes.
Q: What is a dynamic topology change and when does it occur?
A: The reason for having dynamic topology changes is energy saving. Cells whose energy efficiency features are enabled, can be enabled or disabled depending on the network and service demands.

Q: How are dynamic topology changes related to a process that verifies configuration changes?
A: Strictly speaking, dynamic topology changes are configuration changes since the enabling or disabling of a cell requires the change of CM parameters. Hence, they require the same attention as other configuration changes since they may create the same type of problems as presented before.

O4.2: Study and evaluate the impact of topology changes on the process of verifying configuration changes, as well as identify the necessary conceptual changes of a verification process.
Q: How do topology changes impact the process of verification?
A: Turning a cell on or off may have a negative impact on the cell profiling (cf. Definition 3.10). In particular, such an event may induce an incomplete profile (cf. Definition 3.12), which can result in the blame and rollback of necessary CM changes.

Q: Can a verification process issue an appropriate corrective action?
A: A verification process as specified by Definition 3.1 cannot generate an appropriate corrective action in the case of dynamic topology changes. The reason is the trigger: it is not a configuration change that leads to an anomaly but the unexpected entering or leaving of UEs.

Q: What are consequences of dynamic topology changes on the process of verification?
A: It may deploy inappropriate or suboptimal corrective actions and even cause the process of verification to rollback changes not harming performance. In addition, the presence of dynamic topology changes may result in a weak corrective action plan (cf. Definition 3.13).

Chapter 4

Related work

This chapter is devoted to the work related to the verification concept that is presented in this thesis. As stated in Chapter 1, a verification strategy spreads over different areas and addresses additional questions besides those known from anomaly detection and diagnosis. Chapter 3 gave a detailed overview of those problems and issues which have to be solved by verification strategies.

However, a concept that is spread over several areas results in the related work to originate from more than a single research field. Concretely, the concept presented in this thesis has similarities with approaches from three categories: pre-action analysis, post-action analysis, and post-action decision making. The pre-action analysis category (cf. Section 4.1) consist of approaches that concentrate on the avoidance of potential conflicts, and therefore also anomalies, when making changes to configuration parameters. The post-action analysis category (cf. Section 4.2) is comprised of approaches that bring into focus the discovery of anomalies as well as the profiling of the network behavior. The last category, i.e., post-action decision making (cf. Section 4.3), includes methods that focus upon the remedy of changes harming performance and the provision of corrective actions. The topics that are described in this chapter are primarily focused on mobile communication networks, however, also consider areas that may experience similar problems as those introduced in Chapter 3.

4.1 Pre-Action Analysis

As the name suggests, pre-action analysis deals with the avoidance of all potential conflicts, anomalies, and collisions that may occur when the network is operating. In general, it can be distinguished between approaches that implement avoidance mechanisms at design-time, i.e., during the process when automation mechanisms are set up, and approaches that focus on the avoidance at run-time. In the latter case, strategies are used that prevent conflicting actions from being deployed.

4.1.1 Conflict Avoidance at Design-Time

Preventing undesired network behavior *starts* at designing a reconfiguration processes, e.g., online or offline SON approaches (cf. Sections 2.3.2 and 3.1.1), in such a way that they minimize the likelihood of conflicts while changing CM parameters. In the upcoming sections, such design methods are outlined and their shortcomings are discussed.

4.1.1.1 Harmonization Strategies

Within the SOCRATES project [KAB+10] the idea of heading harmonization has been introduced. It is presented as a high-level goal to avoid configuration conflicts by harmonizing the SON function execution. In particular, the idea is to align the policies of the deployed SON functions in a way they do not produce conflicting configuration changes. Such a strategy, though, is limited by the ability to foresee conflicts and predict their impact on the network.

4.1.1.2 Co-Design Approaches

In a similar way, SON function co-design aims to identify potential conflicts and provide a conflict-free SON operation [HSS11, Ban13]. Concretely, it defines a triple of symptoms, possible problems, and triggers of configuration changes. The trigger is a function which solves a problem, like poor coverage, and changes CM parameters accordingly. As a result, a function may trigger other effects like a high number of handover drops, which are identified as symptoms.

There are numerous issues that are not addressed by SON function co-design. First, it is impossible to specify all potential symptoms, that is, predict the consequences of CM changes on the network performance. Second, the list of symptoms is usually narrow which means that a lot of configuration changes will be associated with the same symptom. Hence, the only possible way to prevent conflicts is to sequentialize the action execution process which is very similar to the problem of having a completely serialized corrective action execution plan (cf. Section 3.2.2).

4.1.2 Conflict Avoidance at Run-Time

This section is devoted to the mechanisms that implement run-time conflict avoidance. In general, there are two classes between which we can distinguish. On the one hand, we have approaches that coordinate the action execution at run-time, i.e., they decide whether the requested parameter changes will result in a conflict and whether they are going to negatively impact the performance of the network. If they do so, CM change requests can be rejected. On the other hand, there are methods that reconfigure SON functions in a way that they follow a common objective. By making them to operate towards a common goal, the probability of getting run-time conflicts and collisions may be reduced. Also, there is a third class, namely methods that dynamically adapt the

resource allocation within the network. In the upcoming sections, those strategies are discussed in more detail.

4.1.2.1 Adaptive Action Execution

The concept of pre-action SON coordination [BRS11, RSB13] can be seen as an alternative strategy to a verification strategy. As stated in Section 2.3.2.3, SON coordination is focused on the analysis of actions *before* they are executed. Concretely, it makes use of rules and policies that enable the detection of *known conflicts* between active SON functions. For example, two functions changing the same CM parameter must not run at the same time. If they do so, conflicts may occur and the performance of the network may decrease.

Since pre-action coordination tries to prevent conflicts, rather than resolving them, it can be seen as pessimistic approach since it hinders conflicting functions from getting active. Furthermore, pre-action coordination is not obligated to assess the network performance after the deployment of CM changes. In contrast, a verification process (cf. Definition 3.1) can be categorized as a optimistic approach as it generally allows configuration changes to be applied and only those harming performance are rolled back.

The main problem pre-action coordination is facing is the inability of predicting all potential conflicts. It is generally difficult to foresee how configuration changes are going to impact network performance, e.g., if they are going to induce anomalies.

The approach presented in [CAA13] goes one step further. It presents a coordination method for autonomic configuration changes in mobile communication networks. Instead of relying on pre-defined rules and policies, a mathematical model is proposed which specifies the interaction of SON functions. The authors model SON mechanisms as control loops and describe the system as an Ordinary Differential Equation (ODE). Furthermore, they analyze the stability of the network by using the Lyapunov stability theorem. Should the stability of the network decrease, they enable coordination which has the task of guaranteeing the flawless operation of the SON mechanisms. Unfortunately, the approach does not completely solve the issues that emerge when rolling back changes.

4.1.2.2 Adaptive Configurations of SON Mechanisms

In a mobile network, the complexity of coordinating SON functions at run-time increases with the number of active functions. In the worst case scenario, a coordinator would need to foresee a high number of possible conflicts which will lead to the problems harmonization and co-design approaches are already experiencing (cf. Sections 4.1.1.1 and 4.1.1.2). Furthermore, the increasing complexity of the coordination problem also impacts a verification process as the likelihood of taking suboptimal decisions increases as well. Consequently, a verification process would need to analyze numerous configuration changes and may potentially face uncertainties while rolling back changes, e.g., verification collisions (cf. Definition 3.3).

This problem has been identified in [MATTS15] where the authors propose a framework for Cell Association Auto-Configuration (CAAC). The idea is to generate cell associations which can be updated at run-time. Concretely, an association changes the spatial scope of active SON functions by tweaking their configuration parameters. Cell associations are generated by taking into account the current cell capabilities, information about the network topology, manually defined policies, as well as expert knowledge. Nevertheless, function activities that actually harm performance are not addressed.

The idea of dynamically adapting the configuration of SON mechanisms has also been identified in [LSH16]. The authors introduce an approach that generates SON objective models, i.e., configuration sets that change the parameters of SON functions based on objectives as given by the human operator. Concretely, an objective defines the desired range of the KPIs of interest, e.g., a call drop rate below 5%. A model is computed by a simulation environment that replicates the real network and determines the impact of all possible function parameter values on those KPIs.

Although the approach is focused on reaching certain KPI objectives, it indirectly addresses some of the issues that are targeted by verification strategies. By setting up SON functions to work towards the same goal, the probability of facing run-time conflicts can be minimized. Therefore, the likelihood a SON function to induce a degradation may be minimized as well. However, the capability of generating appropriate objective models also depends on the accuracy of the used simulation model as well as the complexity of the network. As stated in Section 3.1.1, an inaccurate or incomplete model may lead to wrong assumptions about the network and, therefore, to suboptimal optimization decisions.

4.1.2.3 Adaptive Resource Allocation

Besides the mobile network area, there is also the Wireless Local Area Networks (WLANs) area which is commonly employing mechanisms that follow verification paradigms, as discussed in Section 3.1.2. For instance, in [RPM05] a technique is introduced for the assignment of frequencies for WLANs by constructing a so-called interference graph. The suggested solution is based on graph coloring, in particular on minimum vertex coloring. The vertexes of the graph represent the wireless access points whereas the edges connect two access point that would interfere with each other. In addition, a set of maximum possible colors is defined by collecting the number of channels available to the access points.

The problem of solely relying on graph coloring is introduced later in this thesis (cf. Section 6.4). Minimum vertex coloring is not able to solve an over-constrained problem, that is, it is impossible to find a solution if the total number of available colors is insufficient. As a result, a suboptimal parameter choice can be made.

4.2 Post-Action Analysis

Approaches that fall within the post-action analysis class are focused on the assessment of already executed actions. Here, a differentiation is made between strategies that follow anomaly detection and diagnosis paradigms, and methods that focus on the assessment of scope changes. The discussed strategies make use of algorithms that are known from other areas besides mobile networks. Even though some of them are applied on mobile communication networks, they are generally applicable to various other environments and areas.

4.2.1 Degradation Detection and Diagnosis Strategies

In [CCC+14b], an anomaly detection and diagnosis framework has been proposed. It attempts to verify the effect of CM changes by monitoring KPIs exported by the network. In particular, the authors perform anomaly detection and diagnosis on a group of selected cells, i.e., the assessment is not made on the level of an individual cell. Those sets of cells are processed in two steps. During the first step, anomalies are detected by using topic modeling. In literature, topic modeling is known as a statistical model that performs cluster training and formation based on a common criteria. Initially, it has been used to determine the probability distribution of words in documents. During the second step, diagnosis is applied. Concretely, the authors make use of Markov Logic Networks (MLNs) which is an approach for probabilistic reasoning that uses first-order predicate logic.

In [SN12, Nov13] an anomaly detection and diagnosis framework for mobile communication systems is proposed. The authors have developed a framework that analyses performance indicators generated by NEs, observes them for anomalous behavior and suggests a corrective action to the operator. Typical counters are the number of successful circuit or packet switched calls. The suggested system consists of three main building blocks: a profile learning, an anomaly detection and a diagnosis module. The profile learning module analyzes historical data and learns all possible realizations of the normal network operation. The normal network operation itself is represented by profiles which describe the usual (faultless) behavior of KPIs. The anomaly detection module monitors the current network performance and compares it to the profiles. Should a significant difference be detected, the diagnosis module is triggered to identify the possible cause. The diagnosis module consist of a knowledge database fed with fault cases by the operator. Furthermore, a performance report containing the suggested corrective action is provided to the operator who can optionally provide feedback to the system for the purpose of improving the diagnosis capabilities.

In [BHO16], a big data system and machine learning approach is introduced. The presented system analyses the performance of the RAN in LTE, learns the baseline behavior of cells along different dimensions. Concretely, it considers multiple dimensions that are comprised of PM counters, KPIs, configurations and radio setting information.

The baseline behavior is specified by so-called signatures that associate configuration parameters with performance indicators. They represent performance models along the dimensions which are selected based on expert knowledge.

The authors of [BKKS16] propose an anomaly detection algorithm whose primarily task is to discover unusual KPI values in noisy data. It is referred to as a density based spatial clustering algorithm. It makes use of a sliding window which is used for the grouping of the found anomalies. The grouping itself is done per NE and a decision is made whether the observed entity has degraded. However, an interesting fact about this paper is the statement that false positive anomalies are not seen as a problem as they are usually caused by badly selected input data. As stated in Section 3.2.1, false positives may also emerge when numerous functions try to reach their optimization goal. Furthermore, neglecting false positive anomalies increases the likelihood of having numerous verification collisions and an over-constrained corrective action plan, as given by Definitions 3.3, 3.4, and 3.7.

In [GNM15], an anomaly detection technique for cellular networks has been introduced. It is based on the extended version of the incremental clustering algorithm Growing Neural Gas (GNG) which partitions the input data into smaller groups that show a similar behavior. The GNG approach itself [Fri95] has been introduced for anomaly detection in real-time environments. The presented method is able to identify unusual behavior in the time domain. An example is a cell remaining a whole week in a state that represents the weekend. The presented approach, however, does not consider the verification collision problem.

In [NG15], an anomaly detection technique for cellular networks has been introduced. It is based on the incremental clustering algorithm GNG which partitions the input data into smaller groups. Those groups represent sets of input data that have similar characteristics. The presented method is referred to as Fixed Resolution Growing Neural Gas (FRGNG) and targets the problems of representing the input data as well as determining when to stop collecting PM data. However, the solution does not address problems like resolving verification collisions, eliminating weak ones, as well as solving an over-constrained verification collision problem.

The authors of [GAMK$^+$16] introduce a methodology for the design and evaluation of self-healing concepts in LTE. According to them, the main challenge experienced by self-healing approaches is the difficulty in knowing the effects of each fault cause on the PM data. They also state that it is hard to get labeled cases from real mobile networks, i.e., faults associated with symptoms. The main reasons are the complexity of the network as well as the high variety of CM and PM data. In addition, they mention that even if a problem is solved, the real cause usually remains unknown and may, therefore, appear again in the future. As a consequence, the ability to provide an appropriate corrective action becomes challenging as well. For this reason, they propose a scheme that is comprised of three aspects: the building of a fault model, the definition of cause-symptom relations, and designing a diagnosis component which is fed with the collected information. The

modeled fault causes are excessive uptilt or downtilt, cell power reduction, coverage hole, interference and mobility. The symptoms for assessing potentially problematic cells are the most relevant KPIs, e.g., handover success rate, RSRP, RSRQ, SINR, and throughput. The diagnosis system is either completely rule-base or based on fuzzy logic. Nonetheless, the ability of the system to determine the root cause depends on the accuracy of the fault model as well as the ability to specify the relevant symptoms.

4.2.2 Scope Change Assessment

The problem of detecting anomalies while having dynamic scope changes is discussed in [CCC+14a]. The authors propose an ensemble method, i.e., a learning algorithm that constructs a set of classifiers which are used to classify data points and to improve the predictive capabilities. They distinguish between neutral and non-neutral KPIs which is very similar to the KPI categories as defined by the concept of SON verification (cf. Section 5.3.1). Neutral KPIs (e.g., cell throughput) are used for assessing dynamic scopes whereas non-neutral ones (e.g., handover drop rate) are considered by the degradation detection mechanism [CLN+14] . The approach itself is based on a Hierarchical Dirichlet Process (HDP) which utilizes stochastic gradient optimization that allows the training process to evolve over time. Moreover, it is adapted to use both KPIs types.

The problem of addressing dynamic topology changes caused, for instance, by energy saving mechanisms has been discovered in other areas besides mobile communication networks. One example are Self-optimizing Wireless Mesh Networks (SWMNs). The authors of [CSW+15] introduce a wireless backhaul approach that addresses three general issues. As discussed in Section 3.2.5, the authors have identified that dynamic network topology changes can impact the existing traffic. For instance, when new nodes are added or when nodes are removed by putting them into sleep mode, they may cause unexpected changes in the assumptions about the network. For this reason, the authors consider additional aspects like the resilience state of the nodes, i.e., how to avoid a single point of failure and how to reduce the overall impact of topology changes. However, the problem that is not considered is incomplete profiling, as given by Definition 3.12.

4.3 Post-Action Decision Making

Post-action decision making is dealing with the scheduling and deployment of corrective actions which, as the various references later describe, is a topic known from several research areas. In this section, we can distinguish between methods following self-healing paradigms, and such that are based on coordination principles. The presented principles and paradigms are known from mobile communication networks and wireless mesh networks.

4.3.1 Troubleshooting and Self-Healing Approaches

In [FTS+14a, FTS+14b], a concept for operational troubleshooting-enabled SON coordination is given. If a SON function encounters a problem and has at the same time a high assigned priority by the SON coordinator, it may block other functions from getting active. Thus, it can monopolize the network which can result in an unusable SON that is trapped in a deadlock. In such a case a SON function may need assistance by another SON function. The approach proposed by the authors is a SON troubleshooting function that analyses whether SON functions are able to achieve their objectives. If a function encounters a problem that hinders it from achieving its task, the troubleshooting function may trigger another one that may provide a solution to the problem.

In [KAB+10], ideas have been developed about how undesired behavior can be detected and resolved in a SON. The authors introduce a so-called Guard function whose purpose is to detect unexpected and undesirable network performance. They define two types of undesirable behavior: oscillations and unexpected absolute performance. Into the first category usually fall CM parameter oscillations. The second category includes unexpected KPI combinations such as a high RACH rate and low carried traffic. The Guard function itself follows only the directive of the operator defined through policies, i.e., it requires knowledge about expected anomalies. In case such a behavior is detected, the guard function calls another function, called an alignment function, to take countermeasures. The latter one is further split into two sub-functions: an arbitration and an activation function. The first one is responsible for the detection and resolution of conflicting action execution requests. The second one is responsible for enforcing parameter changes, undoing them in case the Guard function detects an undesired behavior, and even suggesting SON function parameter changes. Despite the given ideas, no detailed concepts are provided. Furthermore, the questions of how to select the verification scope and how to address the verification collision problem remain unanswered.

The authors of [BMQS08] introduce a policy-based self-healing system for mobile communication networks. The system itself combines three organizational structures, namely a centralized, a distributed, and a hierarchical one, mainly to allow the choice of the most suitable task distribution over the network. In order to determine the behavior of an NE, the authors specify several models which are used for stability control and dynamic policy conflict resolution. Furthermore, the introduced system makes use of Bayesian networks to detect and also to predict whether changes in the network will negatively impact the performance. It also uses its policies to generate an appropriate event for a given network state. Those policies are condition-action rules which are specified by the human operator.

The idea of rolling back changes based on the outcome of an anomaly detection process has also been introduced. Heckerman et al. [HBR95] developed a procedure that tries to determine not only the most likely cause for a malfunctioning device, but to assemble an action plan for repair. This optimal troubleshooting plan is a sequence of observations and repairs that at the same time minimizes the expected costs. The authors make use

of decision theory as well as Bayesian networks for generating the plan. Their work presents the concept from a theoretical point of view, thus, making it applicable to various research areas.

In [PB98], the authors introduce the idea of modeling the scheduling of pending actions as a constraint optimization problem. The authors propose generalizations of non-preemptive constraint propagation techniques to preemptive ones. Furthermore, they present the concept of generalizing those techniques to so-called mixed problems which allow certain activities to be interrupted whereas others remain non-preemptive. The paper gives a theoretical overview of the concept.

4.3.2 Coordination-Based Approaches

The idea of rolling back already executed configuration changes in SONs has been introduced in [RSB13]. The authors discuss the idea of extending the default set of coordination actions. Besides generating an Acknowledgment (ACK) or a Non-Acknowledgment (NACK) upon a CM change request of a SON function, a SON coordinator could also be allowed to undo already acknowledged requests. In particular, SON function changes are rolled back only if another, higher prioritized and conflicting function becomes active within the same area and at the same time. Thereby, the SON function priorities as well as the coordination policy are taken into consideration in order to make such a decision. The authors have recognized the risks that arise when rolling back changes and, therefore, propose a coordination mechanism that buffers pending configuration requests and also dynamically changes the priorities of the active SON functions. The main target is to limit the number of rollback actions by observing the collected requests. However, the mechanism takes into consideration only coordination properties which, as stated in Section 4.1.2.1, are only able to prevent and resolve *known* conflicts.

In a similar way, in [JFG+13] a coordination concept is introduced which besides conflict detection and priority handling also performs conflict resolution and even learns those being repeated over time. In addition, it is also proposed to undo actions that the coordination framework permitted for execution. The general idea is to accelerate the process of reaching an optimization goal by monitoring network data, and then, if required, to ask previously active functions to rollback their decisions. Furthermore, a possible target configuration may be suggested by the coordination entity. Unfortunately, there are numerous issues that may emerge when following such a strategy. First, it does not directly address degradations that are caused by the activity of the SON functions, i.e., the rollback should speed up the optimization process instead of returning the network to a previous stable state. Second, the assumption that a SON function supports rollbacks, i.e., it is able to handle undo requests, may not be always true. Some SON functions are stateless, i.e., they do not keep track of their changes. Third, executing an undo because the PM data is not showing the expected values leads to the same problems as the rollback generated due to degraded cell performance. Section 3.2 discusses those in detail.

4.4 Summary

In this chapter, the work that is related to the SON verification concept has been presented. The process of verifying changes in the network is spread over several research areas which results the work that is related to the concept of this thesis to originate from more than one research field. In general, those fields can be grouped into pre-action analysis, post-action analysis, and post-action decision making. Throughout this chapter, representatives of each of those groups have been introduced and their disadvantages have been discussed. Moreover, aspects from mobile networks as well as other research areas have been presented. The analysis of those disadvantages contributes to the following research objective:

O1.1: Investigate and study the disadvantages and weaknesses of approaches that verify network operations.
Typically, approaches that fall within the pre-action analysis class (cf. Section 4.1) are only able to foresee known conflicts and, therefore, prevent only a limited set of fault cases. Moreover, they are incapable of predicting the impact of configuration changes on PM data which further limits their capabilities to prevent undesired network states.

Representatives of the second category, i.e., post-action analysis (cf. Section 4.2), are mainly focusing on the detection of unusual and abnormal cell behavior. As a result, they usually do not consider the deployment of corrective actions and, therefore, neglect the issues that emerge when rolling back configuration changes, as presented in Section 3.2.

Strategies that perform post-action decision making (cf. Section 4.3) usually provide a limited set of corrective actions since they either rely on the interaction of the active SON functions or do not have enough knowledge about the network to provide a single action, which would circumvent the problems that emerge when we start to sequentially rollback already deployed CM changes.

Part III

The Concept of SON Verification

Chapter 5

Verification Terminology and Analysis

This chapter is devoted to the structure, properties and attributes of the process of SON verification. It starts with Section 5.1 which gives an overview to the process itself. Concretely, it describes the sub-processes, i.e., CM and topology verification, as well as the phases the verification process is comprised of. The subsequent sections present the steps of the verification process. Section 5.2 discusses the question how to define the scope of verification, i.e., how to partition the network into areas that are analyzed by the process. Section 5.3 specifies the assessment phase, the KPI categories as well as the KPI types relevant on the one hand for CM verification, and on the other hand for topology verification. Section 5.4 focuses on the corrective actions that are generated by the process of SON verification. Section 5.5 focuses on the specification of the verification collision problem, whereas 5.6 introduces the verification time windows.

Finally, the chapter concludes with a discussion about the similarities between SON verification and other concepts following its principles (cf. Sections 5.7 and 5.8), as well as a summary (cf. Section 5.9) that outlines the contributions to the research objectives of this thesis.

Published work This chapter is based on already published work and provides a more comprehensive and detailed description of the verification process structure and terminology. The process of SON verification is specified by journal paper [TATSC16c] and conference paper [TATSC16b]. It should be noted that in order to present the concept in a uniform way, the notation as well as the terminology have been adapted.

5.1 The Process of SON Verification

In general, the SON verification process operates in two phases, as Figure 5.1 outlines. At first, we have the observation phase during which PM and CM data is collected as well as the statistical relevance of the PM data is determined. Furthermore, the type of verification is selected. On the one hand, we have CM change verification, i.e., we assess

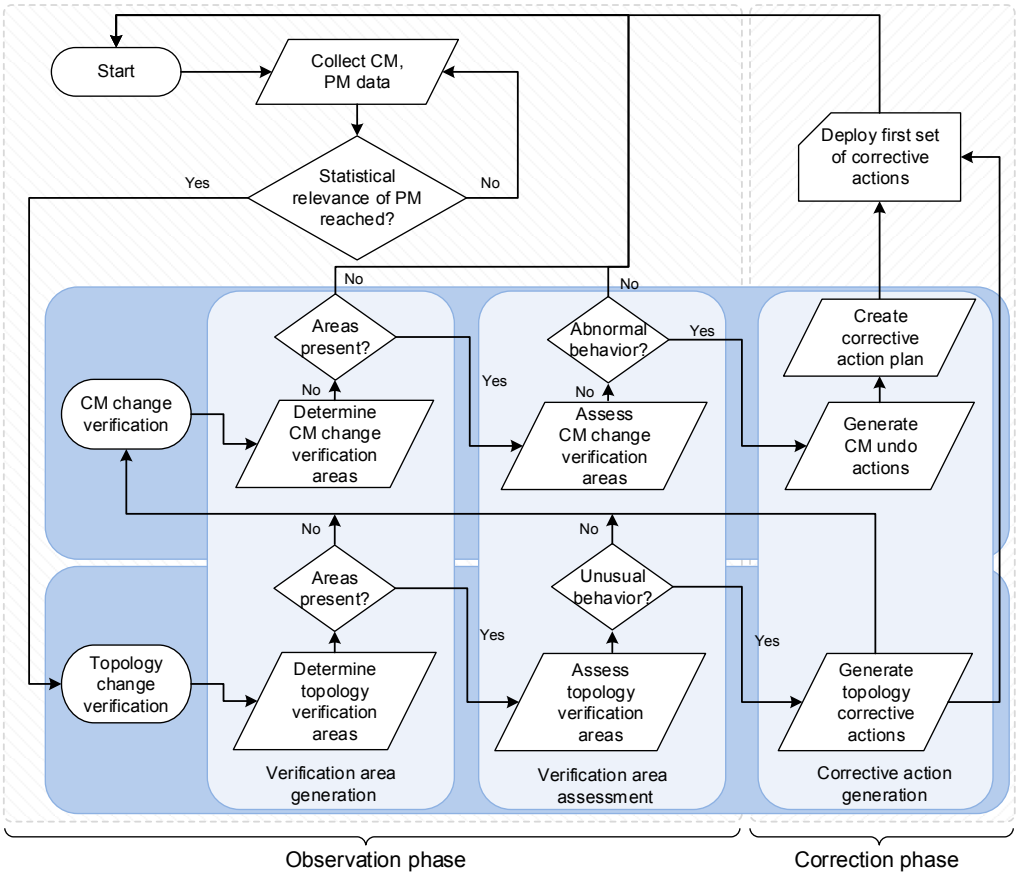


Figure 5.1: Overview of the verification process

the impact of configuration changes on the network performance that have been made in the past. On the other hand, there is a so-called topology change verification whose task is to monitor the availability of the cells, in particular, whether they have been enabled or disabled due to UEs movements (cf. Section 3.2.5).

After selecting the verification mode, the network is partitioned into sets of cells, called *verification areas*. Then, the performance of each area is assessed by triggering an anomaly detection algorithm which identifies those that are abnormally performing.

Afterwards, a transition to the correction phase is made during which *corrective actions* are generated. It the case of CM change verification, a decision is made whether to accept or to revert some of the configuration changes back to a previous stable state. The changes most likely being responsible for causing a degradation are combined into a CM undo action. In the case of topology verification, the choice is made if cells should be activated or deactivated. Finally, a corrective action plan is created and the first set of actions is executed, which is later assessed by the verification process, i.e., the procedure is restarted.

5.2 Verification Areas

First of all, let us denote the set of all cells in the network as Σ. In addition, let us define the scope of the verification process as a set of cells Σ^M, where $\Sigma^M \subseteq \Sigma$. Here, the set Σ^M consists of all cells that are monitored by the verification procedure. This set is further divided into nonempty subsets $\{\Sigma_1^{M'}, \ldots, \Sigma_i^{M'}\}$, which is also referred to as the *verification scope fragmentation*.

Definition 5.1 (Verification scope fragmentation)**.** If we denote the fragmentation of the cell set Σ^M as $\mathcal{P}^\Sigma$, i.e., $\mathcal{P}^\Sigma = \{\Sigma_1^{M'}, \ldots, \Sigma_i^{M'}\}$, the following conditions must hold for $\mathcal{P}^\Sigma$:

- $\emptyset \notin \mathcal{P}^\Sigma$
- $\bigcup_{\Sigma^{M'} \in \mathcal{P}^\Sigma} \Sigma^{M'} = \Sigma^M$
- for any two sets $\Sigma_1^{M'} \in \mathcal{P}^\Sigma$ and $\Sigma_2^{M'} \in \mathcal{P}^\Sigma$: $\Sigma_1^{M'} \neq \Sigma_2^{M'}$

Furthermore, we may have a partition [Bru09] of the set Σ^M, which is referred to as a *strict verification scope fragmentation* (see definition below).

Definition 5.2 (Strict verification scope fragmentation)**.** A fragmentation of the verification scope $\mathcal{P}^\Sigma = \{\Sigma_1^{M'}, \ldots, \Sigma_i^{M'}\}$ is called strict if and only if the intersection of every two cell sets $\Sigma_i^{M'}, \Sigma_j^{M'}$ is empty, i.e., following condition must hold:

- if $\Sigma_1^{M'}, \Sigma_2^{M'} \in \mathcal{P}^\Sigma$ and $\Sigma_1^{M'} \neq \Sigma_2^{M'}$ then $\Sigma_1^{M'} \cap \Sigma_2^{M'} = \emptyset$

Each of the cell sets $\{\Sigma_1^{M'}, \ldots, \Sigma_i^{M'}\}$ represents a *verification area*, which is specified in the following way:

Definition 5.3 (Verification area)**.** A verification area is a subset of cells $\Sigma^{M'} \subseteq \Sigma$, where Σ is the set of all cells. Each verification area is represented as a graph $G^{M'} = (V^{M'}, E^{M'})$. The vertex set $V^{M'}$ represents the cells whereas the edge set $E^{M'}$ the neighbor relations between cells. Furthermore, $G^{M'}$ has the property of being connected, i.e., there is a path between every pair of vertexes.

5.2.1 Verification Areas Based on CM Changes

When we verify CM changes, a verification area is set to include the reconfigured cell, also referred to as the *target cell*, and a set of cells surrounding it, called the *target extension set*. In particular, the target extension set consists of cells that have been possibly impacted by the reconfiguration of the target cell. Typically, they are selected based on the existing neighbor relations, for instance, by taking all first degree neighbors of the reconfigured target. Note that two cells are called neighbors when they have an active neighbor relation between each other that allows them to handover UEs. Figure 5.2

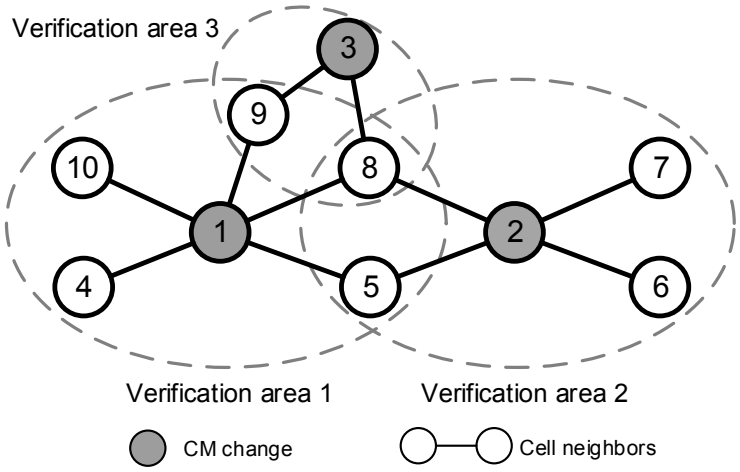


Figure 5.2: Verification area formation in case of CM changes. Each of the three areas is formed by taking the reconfigured cell and its first degree neighbors.

shows an example. A verification area is formed around the cell that has been reconfigured and includes the first degree neighbors.

Furthermore, in case of SON activities, we may define the verification area as the impact area of the SON function that has actually made the change. As discussed in Section 2.3.2.3, the impact area is an important property when coordinating SON actions since it specifies the spatial scope within which a SON function modifies CM parameters and where it takes its measurements from. This property is useful when verifying CM changes since it provides additional contextual information.

A verification area may also be subject to the location of the cells. For instance, if a cell is part of dense traffic or known trouble spots [Eri12], it may join a certain verification area even if it was not initially supposed to do so.

5.2.2 Verification Areas Based on Topology Changes

When we verify topology changes the process of generating verification areas becomes slightly different. First of all, we have to take into account the trigger that has led to the enabling or disabling of a cell. As discussed in Section 3.2.5, the most common reason why cells are put into sleeping mode during their operation is energy saving [3GP10]. That is, the objective is to conserve energy by switching on only those entities that are required, i.e., cells that are needed to meet the network and service demands. Note that in most cases those entities are small cells and not macro cells, as the deactivation of a macro cell may induce a coverage hole. In this thesis, such small cells are referred to as *on-demand* whereas the entities that remain always switched on are called *static* cells.

The afore-mentioned demands may change when numerous UEs enter or leave the network. As a result, a verification area is formed around static cells as they may show an anomalous behavior like an unusually high load or throughput. Figure 5.3 shows a

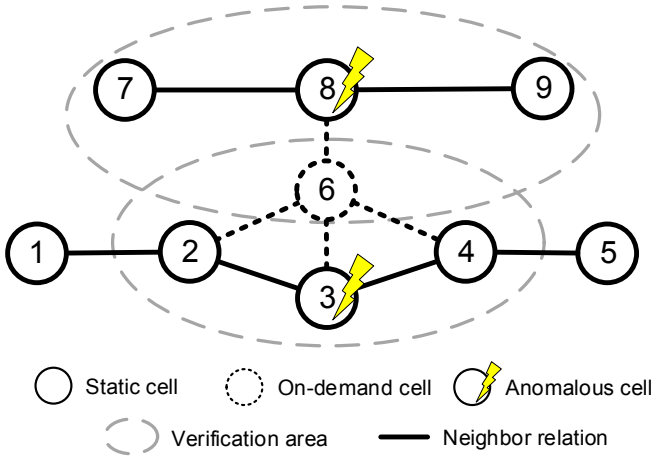


Figure 5.3: Verification area formation in case of dynamic topology changes. An area is formed around anomalous static cells which can potentially handover users to on-demand cells.

permissible verification area selection. An area is formed around every anomalous static cell. Compared to verification areas formed around CM changes, a topology verification area also includes disabled on-demand cells. Those cells are required for the provision of topology corrective actions (cf. Section 5.4.2).

5.3 Verification Area Assessment

During this phase, the performance of each verification area $\{\Sigma_1^{M'}, \ldots, \Sigma_i^{M'}\}$ is assessed. In SON verification, an anomaly detection algorithm is used for that purpose. Thereby, cell KPIs are analyzed and how the network should usually behave is specified. It should be noted that the latter one is known as profiling, which is introduced in Section 3.2.5.

Once the cell performance considerably differs from all learned states of the normal network operation, a verification area is marked as potentially anomalous and is later considered while forming and processing the corrective action plan (cf. Definition 3.4). In the upcoming sections, an introduction to the KPIs relevant for SON verification as well as an overview of the different profile types is given.

5.3.1 KPI Categories

Similarly to anomaly detection in mobile networks [Nov13], or network management in general, there are three KPI categories between which we can distinguish:

- Success, KPIs, rates and counters
- Failure KPIs, rates and counters
- Neutral KPIs, rates and counters

Examples that fall within the first category are the Handover Success Rate (HOSR) and the Call Setup Success Rate (CSSR), which measure the rates of successfully established handovers and calls, respectively. The second category typically consists of rates like the call drop rate and the number of radio link failures. The last category includes KPIs like the cell load and the cell throughput. As stated in [Nov13], such KPIs are not bound to any success or fault events.

The KPIs that are selected by the verification process can fall within any of the aforementioned categories. In this thesis, they are referred to as *cell state KPIs*, which are further specified as follows:

Definition 5.4 (Cell state KPI)**.** A cell state KPI k is element of $\mathbb{R}$ and is used by at least one of the verification sub-processes, i.e., CM and topology verification. They are required to define the performance state of a cell within a verification area.

5.3.2 CM Verification KPIs and Profiles

Every cell exports KPIs, however, only a certain subset of those plays a role for CM verification. They are referred to as *Configuration management verification KPIs (CKPIs)*[1] and are defined as follows:

Definition 5.5 (CKPI)**.** A CKPI $k^{\perp} \in \mathbb{R}$ is an element of a vector $\mathbf{k}^{\perp} = (k_1^{\perp}, \ldots, k_n^{\perp})$, also called a CKPI vector. It is required to define the state of a cell $\sigma \in \Sigma$ with respect to CM verification. The collection of all CKPI vectors is denoted as $K^{\perp} = \{\mathbf{k}_1^{\perp}, \ldots, \mathbf{k}_{|\Sigma|}^{\perp}\}$, where $\mathbf{k}^{\perp} \in \mathbb{R}^{|\Sigma|}$ and Σ is the set of all cells.

There are numerous examples of how to select those KPIs. For example, if we are interested in the handover performance, we may consider indicators like the HOSR, the handover ping-pong rate, or the rate of too early handover attempts. If the channel quality needs to be assessed, we may include a KPI like the CQI.

Furthermore, the CKPI profile is denoted as $\mathbf{p}^{\perp} = (p_1^{\perp}, \ldots, p_n^{\perp})$ and is defined in the same way as given by Definition 3.10. The collection of all CKPI profiles is identified by the set $P^{\perp}$.

5.3.3 Topology Verification KPIs and Profiles

The topology verification process distinguishes between static cells, i.e., such that are never turned off, and on-demand cells, i.e., such that may be turned off during their operation. Let us denote the set of all static cells as $\dot{\Sigma}$ and the set of all on-demand cell as $\mathring{\Sigma}$. Moreover, let $\dot{\Sigma} \cup \mathring{\Sigma} = \Sigma$ and $\dot{\Sigma} \cap \mathring{\Sigma} = \emptyset$, where Σ is the set of all cells. In addition, let us call the KPIs relevant for topology verification *Topology verification KPIs (TKPIs)*[2] and define them in the following way:

[1] CKPIs have also been referred to as dedicated KPIs [TATSC16c].

[2] TKPIs have also been referred to as utilization KPIs [TATSC16b].

Definition 5.6 (TKPI)**.** A TKPI $k^{\vdash}$ is an element of a TKPI vector $\mathbf{k}^{\vdash} = (k_1^{\vdash}, \ldots, k_n^{\vdash})$, where $\mathbf{k}^{\vdash} \in \mathbb{R}^n$. The vector represents the state of a cell $\sigma \in \Sigma$ with respect to topology verification. In the case of a static cell, the vector is denoted as $\dot{\mathbf{k}}^{\vdash} = (\dot{k}_1^{\vdash}, \ldots, \dot{k}_n^{\vdash})$ whereas for on-demand cells the vector is denoted as $\mathring{\mathbf{k}}^{\vdash} = (\mathring{k}_1^{\vdash}, \ldots, \mathring{k}_n^{\vdash})$. The collection of all TKPI vectors is called the TKPI vector space and is denoted as $K^{\vdash} = \{\mathbf{k}_1^{\vdash}, \ldots, \mathbf{k}_{|\Sigma|}^{\vdash}\}$, where $K^{\vdash} \subset \mathbb{R}^{|\Sigma|}$ and Σ is the set of all cells. In addition, $\dot{K}^{\vdash} = \{\dot{\mathbf{k}}_1^{\vdash}, \ldots, \dot{\mathbf{k}}_{|\dot{\Sigma}|}^{\vdash}\}$ and $\mathring{K}^{\vdash} = \{\mathring{\mathbf{k}}_1^{\vdash}, \ldots, \mathring{\mathbf{k}}_{|\mathring{\Sigma}|}^{\vdash}\}$ are the TKPI vector spaces of the static cells $\dot{\Sigma}$ and on-demand cells $\mathring{\Sigma}$, respectively.

Usually, a TKPI falls within the neutral KPI category, i.e., it is not bound to any failure or success events. Examples are the cell load, the cell throughput, and the data traffic that passes through a cell, i.e., KPIs that represent the utilizations state of cells.

In contrast to CM verification, only static cells have a profile. Let us denote the profile as $\dot{\mathbf{p}}^{\vdash} = (\dot{p}_1^{\vdash}, \ldots, \dot{p}_n^{\vdash})$, i.e., a vector whose elements $\dot{p}_i^{\vdash}$ represent values, or intervals of values, that define the usual range of the selected KPIs (cf. Definition 3.10). Moreover, let us call this profile a TKPI profile and denote the collection of all such profiles as $\dot{P}^{\vdash}$. Such a profile specifies the usual behavior of the TKPIs of a vector $\dot{\mathbf{k}}^{\vdash} = (\dot{k}_1^{\vdash}, \ldots, \dot{k}_n^{\vdash})$ when no on-demand cell is operating.

5.4 Corrective Actions

The concept of SON verification distinguishes between two types of corrective actions. On the one hand, there is a CM undo action whose purpose is to set a cell's configuration to previous stable state. It is generated by the CM verification process. Note that its structure is further discussed in Section 5.4.1. On the other hand, a topology corrective action can be generated by the topology verification process. It has the purpose of switching an on-demand cell either on or off. Section 5.4.2 describes it in more detail.

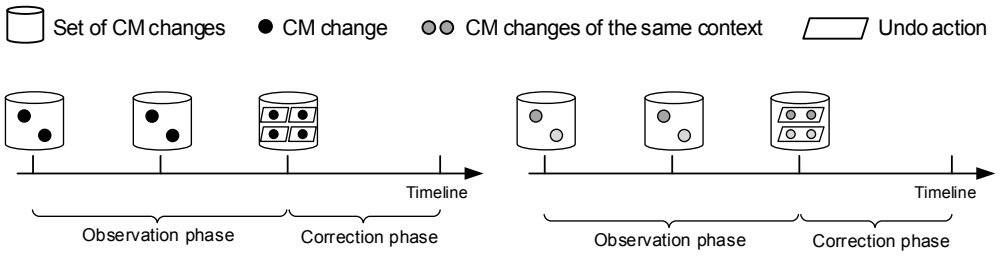
5.4.1 CM Undo Action

Reconfigurations that have led to abnormal cell performance have to be rolled back. For this purpose, CM undo actions are specified which return cell configurations to a stable state. The set of all such actions is denoted as $C^{\perp}$ whereas a single action as $c^{\perp} \in C^{\perp}$. An undo action is generated for the target cell of a verification area, as introduced in Section 5.2. Moreover, it contains the following type of information:

- Identifier of the target cell within the verification area.
- Identifiers of the cells included in the target extension set.
- List of CM parameter values for the target cell.

The list depends on the CM parameter type as well as the origin of the CM change. On the one hand, if we do not have any contextual information about the change, each CM

change that has been marked as potentially bad will be considered as a separate undo action. Figure 5.4(a) visualizes this process. On the other hand, if we are in possession of such information, we can combine some changes into a single undo action. As shown in Figure 5.4(b), in the case of SON function activities, the parameter changes made by a function is considered a single autonomic action which can be rolled back. For instance, if CCO reconfigures the transmission power and antenna tilt degree at the same time, the verification process may undo both if required.



(a) No contextual information about CM changes (b) With contextual information about CM changes

Figure 5.4: Permissible undo actions in the case of four CM changes

Furthermore, in the case of SON activities we may have more than one target cell for the same undo action, i.e., we have a target set instead of a target cell. It comes from the fact that the function area of a SON function may include more than one cell, which means that the changes made to those cells are seen as a single autonomic configuration change. For instance, in Figure 5.2 the undo for verification area 1 and 2 can be combined into one single action if the changes of cell 1 and 2 have been triggered by the same SON function. The MRO function, which optimizes the handover between neighboring cells, is one example. It may not only change the CIO between a cell and its neighbor, but also vice versa. As a consequence, we would need to consider both cells as targets since both have been reconfigured by the same function.

5.4.2 Topology Corrective Action

When verifying CM changes the corrective action in case of an anomaly, e.g., a degradation in performance, is to undo the changes that have caused it. However, when we verify dynamic changes in the network topology we need to do more than that. The simplistic case when a cell is turned on or off by mistake can be corrected by generating an undo action. Unfortunately, if the initial assumptions about the environment change, e.g., a high number of UEs enters the network and causes some cells to be anomalous, we would not be able to generate an appropriate corrective action.

For this reason, we need to extend the corrective action set by adding such that can actively change the network topology. Hence, a verification mechanism would actively decide whether cells have to be enabled or disabled. Such actions are referred to as

topology corrective actions and their set is denoted as $C^{\vdash}$, where a single action is identified by $c^{\vdash} \in C^{\vdash}$.

5.5 Specification of the Verification Collision Problem

Let us recall the verification collision problem, as given by Definition 3.3. Two corrective actions that rollback cell configurations are said to be in collision in case they share a common trigger. This trigger is returned by function $\zeta : C^{\perp} \rightarrow \mathbb{P}(\Sigma) \setminus \emptyset$, where $C^{\perp}$ is the set of corrective rollback actions and Σ the set of all cells. Hence, this function is the starting point for the verification of CM changes.

The concept of SON verification divides the network into verification areas, each being assessed by an anomaly detection algorithm (cf. Section 5.2). An area is a set of cells that includes the reconfigured (target) cell and a set of cells that are potentially impacted by a configuration change. If an area is degraded then an undo action is generated for the target cell. Thereby, the concept of SON verification further specifies the above-mentioned function as follows:

$$\widetilde{\zeta} : C^{\perp} \rightarrow \mathbb{P}(\Sigma^{A}) \setminus \emptyset \tag{5.1}$$

Instead of returning the power set of all cells, the codomain is the power set of all cells being anomalous, also denoted as Σ^{A}. Hence, two undo actions $c_i^{\perp}, c_j^{\perp} \in C^{\perp}$ are said to be in a verification collision if and only if the two verification areas $\Sigma_i^{M'}, \Sigma_j^{M'} \subseteq \Sigma$, for which they have been generated, share common anomalous cells, i.e., $\Sigma_i^{M'} \cap \Sigma_j^{M'} \subseteq \mathbb{P}(\Sigma^{A}) \setminus \emptyset$.

5.6 Time Windows

As shown in Figure 5.1, the verification process operates in two phases, observation and correction. Each phase is represented as a time window which is further divided into one or more time slots. Figure 5.5 visualizes an exemplary 3-slot observation and 2-slot correction window.

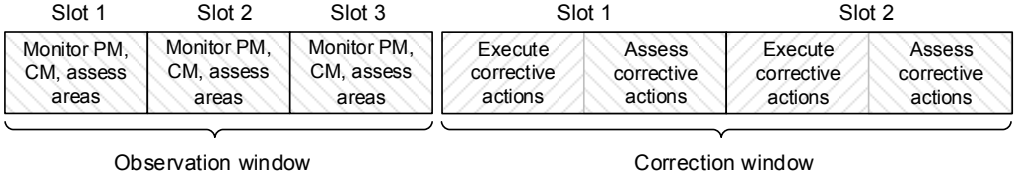


Figure 5.5: An example of a 3-slot observation and 2-slot correction window

5.6.1 Observation Window

During the observation window, the anomaly detection procedure is triggered. That is, PM data is collected, the performance impact of CM changes is assessed, and changes of the

network topology are observed. Moreover, the movements of UEs is closely monitored, and PM data that reflects their behavior is observed. The activity of SON functions is also watched, in particular, function transactions are identified (cf. Definition 3.2).

Furthermore, during the observation window verification areas are formed. As discussed in Section 5.2, they specify the sets of cells that are considered as one entity during the undo decision making process. A comparison between the cell profiles and the current PM reports is also made (cf. Section 5.3).

The duration of an observation window slot depends on the PM data collection frequency, i.e., the PM granularity period (cf. Section 2.2.3). The total number of slots, however, depends on the anomaly detection strategy. For instance, the CM verification process dynamically adapts the observation window based on the reported PM data. The strategy is presented in Section 6.2.

5.6.2 Correction Window

The second window is referred to as the correction window and is used for the deployment of corrective actions as well as for assessing their impact on the network performance. Furthermore, during the correction window verification collisions (cf. Definition 3.3) are resolved, it is determined whether we have an over-constrained corrective action plan (cf. Definition 3.7), and the search for soft verification collisions (cf. Definition 3.8) is triggered. The identification of soft verification collisions is required in the case of an over-constrained verification problem. In addition, the search for weak verification collisions (cf. Definition 3.9) is triggered. The algorithms used to accomplish those tasks are introduced in Chapters 6 and 7.

The length of a correction window slot is reliant on the time it takes until the impact of correction actions becomes visible in the PM data. Hence, it depends in the same way on the PM granularity and the PM data type as the observation window does. The total number of available slots, however, is subject to the environment. For example, in a highly populated area we might have a short correction window since bringing the network performance to the expected range has to be carried out as fast as possible.

5.7 Relation to Feedback Planning

Typically, SON functions are implemented as closed control loops which monitor PM and FM data, and based on their goals change the CM parameters they are responsible for. In Section 2.3.2, a detailed overview of their structure and properties is given. Those functions, however, do not plan in advance the parameter changes they are going to make in the future. Exactly this property is where the main difference between well-known SON functions and the SON verification process lies in.

A SON verification approach can be seen as another class of SON functions, namely such forming a course of actions that have high expected utility rather than having a predefined absolute goal. Those functions plan under uncertainty, either because they

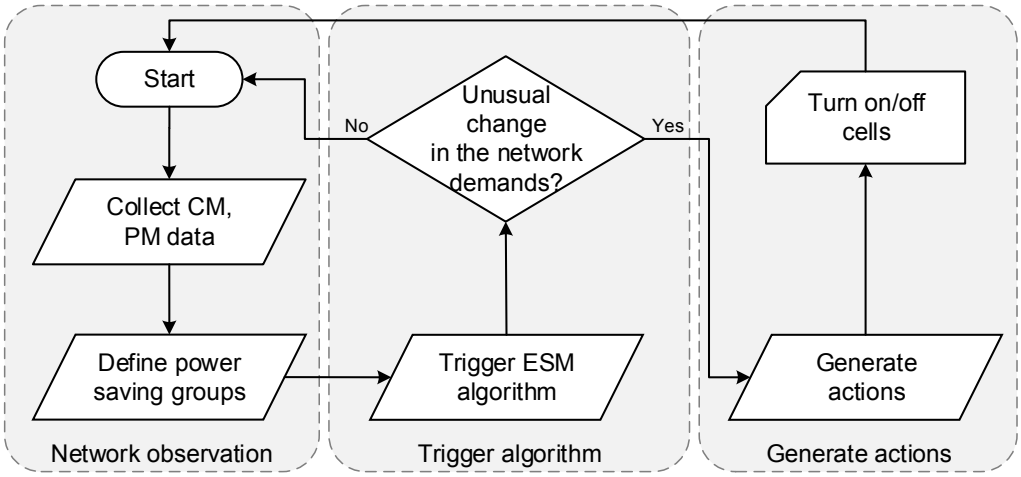


Figure 5.6: Phases of an ESM function

have incomplete information about the world, or because their actions have uncertain effects on the environment. In literature, this type of planning is referred to as Decision-Theoretic Planning (DTP) [BDH99].

A common approach for overcoming uncertainties is to make the actions of the plan depend in some way on the information gathered during execution. This particular strategy is referred to as feedback planning [YL11]. Formally, the problem to find a plan can be represented as

$$C = F(\mathcal{S}, \bar{C}, \tilde{C}), \tag{5.2}$$

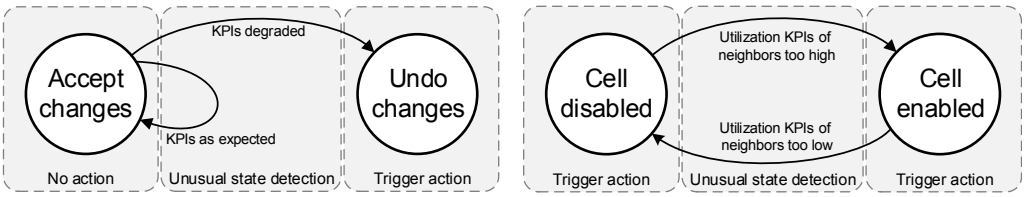
where C represents the action set, $\bar{C}$ the set of actions that have been executed, $\tilde{C}$ the set of actions that are still pending, and $\mathcal{S}$ the current network state.

In terms of SON verification, the set of actions C corresponds to the undo and topology corrective actions, as defined in Section 5.4. The state $\mathcal{S}$ is defined by the outcome of the observation phase, as discussed in Section 5.1. It is characterized by the currently generated verification areas, the current PM reports, and the outcome of the area assessment step.

5.8 Similarities to Energy Saving

The verification process, as introduced in Section 5.1, bears a resemblance to Energy Saving Management (ESM) approaches [MSES12, 3GP10, MAT16]. The similarity to such methods comes mainly from the ability to generate a topology corrective action that switches cells on or off. Furthermore, the trigger is an untypical event caused, for example, by UE groups that have unexpectedly entered or left the network.

However, before going into further details let us take a look at the phases of an ESM function (cf. Figure 5.6). At first, we have a partitioning of the network into so-called



(a) CM change verification (b) ESM function

Figure 5.7: SON verification versus ESM configuration states. The configuration space consist of both functions consists of two possible choices: undoing or accepting changes versus enabling or disabling a cell.

power saving groups. Those groups consists of two types of cells: (1) such that are switched on or off depending on the network and service demands, and (2) cells that are necessary to ensure that there will be always coverage after some cells are put into power saving mode.

Second, each power saving group is assessed by the ESM algorithm. The target here is to search for *unusual* changes in the network demands. For example, some macro cells may be over- or underloaded because numerous UEs have joined or left the network.

Third, within each power saving group a decision is made which cells to enable or disable. Here, several constraints have to be taken into consideration. One would be to keep the number of enabled cells minimal. Another is not to compromise user perception and guarantee high QoS for service requests.

As we can see, there are similarities between the way the verification process operates and the functioning of energy saving methods. Now, let us concentrate on the verification of configuration changes and temporarily forget about the ability to actively enable and disable cells. Similarly to ESM, we need to split the network into sets of cells, assess each one of them, and generate the corresponding actions if required. In the case of CM change verification, we have undo actions that try to put the network performance into the expected range. Hence, we have a twofold decision, as outlined in Figure 5.7(a): either to accept configuration changes that have been deployed, or to reject them and restore past CM settings. The same twofold choice has to be made when ESM operates (cf. Figure 5.7(b)): it either accepts the current state of a cell, or toggles its power saving mode. As a result, the configuration space consist of only two possible values, namely to turn a cell on or off, and not an array of values as it is usually the case for SON functions like MRO or CCO. For instance, the CIO that the MRO function adjusts may accept any real number that is bounded between a predefined minimum and maximum. In addition, having such a simplistic configuration space means that each decision ESM makes for a cell is actually an undo of the change it has made before.

5.9 Summary

In this chapter, the structure and properties of the concept of SON verification have been presented. It is defined as a three step process that splits the network into sets of cells, also called verification areas, assesses each area by using an anomaly detection algorithm, and generates corrective actions for the areas if required. The following answers further highlight this process and outline the research contributions.

O1.2: Model and design a verification process that assesses CM changes as well as changes made in the network topology.
Q: Which are the steps required by the SON verification process?
A: The concept of SON verification starts by partitioning the verification scope. This operation yields a set of verification areas, where each area is a set of cells (cf. Definition 5.1). A verification area has also the property of being a connected graph, in which vertexes represent cells and edges neighbor relations (cf. Definition 5.3). If none of the areas overlap, i.e., share common cells, the fragmentation of the verification scope is referred to as strict (cf. Definition 5.2). Furthermore, there are two types of verification areas: such being formed around the reconfigured cell, and such around cells that can be potentially impacted by a topology change.
Then, each verification area is assessed by an anomaly detection algorithm which identifies those that are anomalously performing. In the case of CM change verification, a test is made whether the performance of the cells of an area has degraded. In the case of topology verification, an evaluation is made whether the utilization of the cells is unusually high or low.
Based on the outcome of the anomaly detection process, a corrective action plan, as given by Definition 3.4, is generated. Moreover, it meets the requirements of being collision-free (cf. Definition 3.5) and gain-aware (cf. Definition 3.6).

Q: Which are the modes of verification?
A: The concept of SON verification supports two modes of operation. On the one hand, there is the CM verification mode, also referred to as the CM verification process, which assesses the impact of already deployed configuration changes on the network performance and rolls back those harming it. On the other hand, the concept defines a so-called topology verification mode. It serves the purpose of observing dynamic changes in the network topology and generating corrective topology actions, i.e., turning cells on or off in case they experience an usually high or low utilization.

Q: Do the verification modes interact with each other?
A: Yes. It is of high importance to make CM verification and topology verification interact, i.e., they have to be informed about each others' outcome. Otherwise, the issues described in Section 3.2.5 may emerge.

Q: Which are the operational phases of the SON verification process?
A: The verification process requires two phases: observation and correction phase. During the observation phase, PM and CM data is monitored, verification areas are formed, and the anomaly detection algorithm is triggered. During the correction phase, the corrective action plan is processed, i.e., all planning-related issues (like verification collisions) are solved.

In this chapter, a contribution was also made to the research objective that investigates the fragmentation of the verification scope. The following answer contributes to the objective:

O2.1: Fragmentation of the network and specifying the scope of verification.
Q: How to model the fragmentation of the network during the process of verification?
A: The verification scope fragmentation is a process during which the cells of interest, i.e., those that are being monitored by the verification mechanism, are split into sets (cf. Definition 5.1). Each of those sets represents a verification area (cf. Definition 5.3) and there are no two verification areas that include exactly the same cells. Furthermore, the fragmentation result is called strict if none of the verification areas share common cells (cf. Definition 5.2), i.e., the resulting cell sets are mutually disjoint.

In addition, the main attributes of the SON verification concept have been presented. The answers to the following questions present those attributes in detail.

O1.3: Identify and specify all necessary attributes and properties of a verification process.
Q: Which are the KPIs of interest for the CM verification process?
A: The KPIs of interest are called CKPIs (cf. Definition 5.5) and fall within the cell state KPI category (cf. Definition 5.4). They define the performance state of a cell and are used for the detection of abnormal cell performance. Also, they can be neutral, success or failure KPIs. Furthermore, a cell's CKPIs form a so-called CKPI vector which is used to determine the behavior of the cell with respect to CM verification.

Q: Which KPIs are of interest for the topology verification algorithm and what do they represent?
A: The KPIs of interest are referred to as TKPIs (cf. Definition 5.6). They represent the state of a cell with respect to topology verification and form a so-called TKPI vector. The collection of all such vectors is called the TKPI vector space (cf. Definition 5.6). Furthermore, TKPIs are usually neutral KPI (cf. Section 5.3.1).

Q: Are there any other corrective action types besides those rolling back CM changes?
A: Yes. There is a second corrective action type that is referred to as a topology

corrective action. It enables or disables cells instead of rolling back their configuration. The action itself is generated by the topology verification process.

Q: Which types of performance indicators are relevant for the verification process?
A: There are three classes of verification performance indicators: success, failure, and neutral KPIs, counters and rates. They are used in the same context as in anomaly detection.

Q: Between which cell types does the concept of SON verification differentiate?
A: In the case of CM verification, there is no differentiation between cells, i.e., all of them are seen as entities that may require verification. However, in the case of topology verification, there are two cell types: static and on-demand cells. Within the first class fall cells that are never turned off during their operation (e.g., macro cells). Representatives of the second class are small cells (e.g., femto and pico cells) that can be enabled or disabled in case the network and service demands change.

In this chapter, also the similarities to DTP, in particular, feedback planning, have been outlined. A comparison between strategies that dynamically change the network topology and the concept of SON verification has been made. The answers to the following questions highlight the resemblances.

O1.4: Study the relation between a verification process and concepts based on decision theory.
Q: Does the verification process relate to feedback planning?
A: Yes. In contrast to well-known SON functions the verification process plans in advance the actions its going to make. Furthermore, it does not have a predefined *absolute* goal and plans under uncertainties. Uncertainties emerge due to incomplete information about the world and the unknown effects of its actions. To overcome those challenges, it makes the actions of the plan depend on the information gathered while it gets processed. This is where the resemblance to feedback planning comes from.

O1.5: Study the relation between a verification process and approaches performing and assessing topology changes.
Q: Does a process that verifies configuration changes relate in any way to energy saving strategies?
A: Yes. The main resemblance comes from the configuration space. In the case of CM verification the outcome is either to accept the recently deployed changes or to reject them by rolling cell configurations back to a previous stable state. The same twofold decision is made by ESM approaches. They either turn a cell on or off, i.e., an ESM action is an undo of the previously executed one. Furthermore, both a verification strategy and an ESM approach operate in three phases, i.e., split the network into areas

(sets of cells), assess each area, and make a twofold decision. Hence, the ability to actively correct topology changes is a task that a verification strategy has to consider. It is a better decision maker as it has a wider view on the network, i.e., monitors other ongoing CM changes and KPIs that can further improve the decision to enable or disable a cell.

Chapter 6

Verification of Configuration Changes

In this chapter, the major topic of discussion is the verification of configuration changes, also known as CM change verification. Figure 6.1 represents the steps required to complete this type of verification. At first, we have the partitioning of the network into verification areas. This step has already been described in Section 5.2 and will not be further discussed here. Then, we have a description of the cell behavior model in Section 6.1. It provides information how to use the KPIs of interest for the specification of the state of a cell with respect to CM verification. In Section 6.2, the anomaly detection procedure is presented. It describes the way of handling fluctuations in the PM data, i.e., how to prevent verification areas from being unnecessarily processed. It makes use of the behavior model defined for each cell. Section 6.3 continues with the MST-based clustering technique that reduces verification collisions. The presented method identifies and eliminates potentially false positive verification collisions by changing the size of the impacted verification areas. Concretely, it removes cells from the initial area selection. It is followed by Section 6.4 which is dedicated to the estimation of the collision grade. The latter one is required for the estimation of the severity of the verification collision problem.

The next two sections, 6.5 and 6.6, discuss the corrective action part, i.e., the generation of a plan, and present the strategy that finds a solution in case the generated plan is over-constrained. Finally, the chapter concludes with a summary (cf. Section 6.7) and the contributions to the research objectives.

Published work This chapter is based on already published work. Paper [TAT16] covers the cell behavior model and the anomaly detection strategy. In [TATSC16a], the clustering technique that is used for reducing verification collisions is described. In [TSC15], the technique that used to estimate the collision grade is specified. Paper [TFSC15] presents the approach for resolving collisions, generating the corrective action plan and the strategy used for handling an over-constrained verification problem. Paper [TATSC16c] presents the complete CM verification process. Note that the notation has been adapted in order to present the concept in a uniform way.

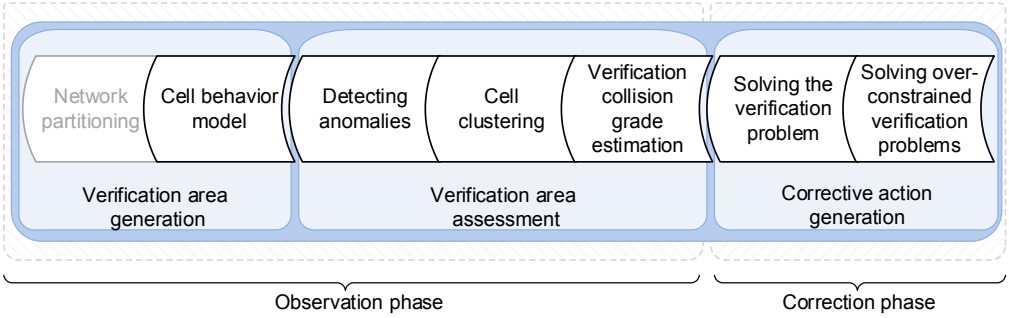


Figure 6.1: Overview of the CM verification process

6.1 Cell Behavior Model

An important part of the CM verification procedure is how we model the behavior of each cell in the network. In particular, we are interested in the impact of configuration changes on the PM data. Nevertheless, to be able to evaluate that we need to represent each cell in a certain way. Here, the behavior of a cell $\sigma \in \Sigma$ is given by its CKPI anomaly level vector, in particular, by the selected CKPI anomaly levels. Note that the term CKPI is specified by Definition 5.5.

Definition 6.1 (CKPI anomaly level)**.** A CKPI anomaly level $a^{\perp} \in \mathbb{R}$ is element of a CKPI anomaly vector $\mathbf{a}^{\perp} = (a_1^{\perp}, \ldots, a_n^{\perp})$, where $\mathbf{a}^{\perp} \in \mathbb{R}^n$. Each $a_i^{\perp}$ represents the deviation of the corresponding CKPI $k_i^{\perp}$ in $\mathbf{k}^{\perp} = (k_1^{\perp}, \ldots, k_n^{\perp})$. Furthermore, for the set of all cells Σ, $A^{\perp} = \{\mathbf{a}_1^{\perp}, \ldots, \mathbf{a}_{|\Sigma|}^{\perp}\}$ is called the CKPI anomaly level vector space, where $|A^{\perp}| = |\Sigma|$.

The CKPI anomaly level vector of a cell is computed by function φ (cf. Equation 6.1) which takes a cell profile $\mathbf{p}^{\perp} \in P^{\perp}$ (cf. Section 5.3.2) and a CKPI vector $\mathbf{k}^{\perp} \in K^{\perp}$ and returns an element of $A^{\perp}$.

$$\varphi \colon P^{\perp} \times K^{\perp} \rightarrow A^{\perp} \tag{6.1}$$

We need to make an important remark here. As stated in Section 3.2.5, not every cell has a valid profile since some can be dynamically turned on or off. As a result, we would not be able to apply the above-mentioned function and compute the anomaly levels. How to do that is the main topic of discussion in Chapter 7 and will not be further explained here. In this chapter, it is assumed that all cells have a complete profile, as given by Definitions 3.10 and 3.12.

Let us give an example for computing $\mathbf{a}^{\perp} = (a_1^{\perp}, \ldots, a_n^{\perp})$. For this purpose, assume that $\mathbf{k}^{\perp}$ consist of the HOSR and CSSR. Here, we may compute each CKPI anomaly level as the z-score [FPP07] of a CKPI, i.e., the distance between the given CKPI value and the sample mean in units of the standard deviation. The samples required to compute that value can be collected separately, e.g., during a training phase. Let us assume that during this phase a cell reports a HOSR of $\{99.9\%, 99.7\%\}$ and a CSSR of $\{99.5\%, 97.4\%\}$.

Also, suppose that the current HOSR and CSSR are 90.2% and 90.0%, respectively. This leads to the following anomaly vector: $\mathbf{a}^{\perp} = (-1.15, -1.13)$, i.e., HOSR anomaly level of -1.15 and CSSR anomaly level of -1.13. Since both are negative, we can state that the current CKPI values are more than one standard deviation below the sample mean.

6.2 Detecting Anomalies

In the previous section, the cell behavior model was introduced as well as an example for computing the cell performance state was given for two particular KPIs, HOSR and CSSR. The example was based on the z-scores of those two indicators which indicated that the current KPIs are more than one standard deviation away from the sample mean. As a consequence, it was stated that the cell reporting those KPIs may be anomalous which would result in the generation of a verification area. As stated in Definition 5.3, the area itself would include that cell as target, as well as cells surrounding it, and will be pushed through the CM verification process.

Nonetheless, an area may be unnecessarily processed since there might be a temporal performance decrease induced by a SON function, as discussed in Section 3.2.1. A function may be executing a transaction (cf. Definition 3.2), as it tries to reach its optimization goal. This may result in fluctuations in the PM data which may lead to wrong assumption about the performance state of a verification area.

Hence, instead of evaluating cells based on the anomaly vectors they report, a Cell Verification State Indicator (CVSI)[1] is defined. As the name suggests, it is computed for each cell and has the purpose of making the verification approach more resistant against PM data fluctuations. The CVSI of a cell is denoted as $\vartheta(\mathbf{a}^{\perp})$, where ϑ is a function with domain $A^{\perp}$ and codomain $\mathbb{R}$, i.e.:

$$\vartheta \colon A^{\perp} \times \mathbb{R} \to \mathbb{R} \tag{6.2}$$

In this thesis, the computation of ϑ is done by applying exponential smoothing, as follows:

$$\vartheta(\mathbf{a}^{\perp}, \alpha)_0 = \psi(\mathbf{a}^{\perp})_0 \tag{6.3}$$

$$\vartheta(\mathbf{a}^{\perp}, \alpha)_t = \alpha\psi(\mathbf{a}^{\perp})_t + (1-\alpha)\vartheta(\mathbf{a}^{\perp}, \alpha)_{t-1},\ t > 0 \tag{6.4}$$

Here, $\alpha \in [0; 1]$ is the smoothing factor, also referred to as the state update factor, whereas $\vartheta(\mathbf{a}^{\perp}, \alpha)_t$ is called the CVSI calculated at time t. It is a simple weighted average of the current observation, denoted as $\psi(\mathbf{a}^{\perp})_t$, and the previous smoothed $\vartheta(\mathbf{a}^{\perp}, \alpha)_{t-1}$. The current observation itself is a function, as defined by Equation 6.5.

$$\psi \colon A^{\perp} \to \mathbb{R} \tag{6.5}$$

It is an aggregation of $\mathbf{a}^{\perp} = (a_1^{\perp}, \ldots, a_n^{\perp})$ that captures the current performance state of a

[1] The term cell anomaly level is used as a synonym for the CVSI.

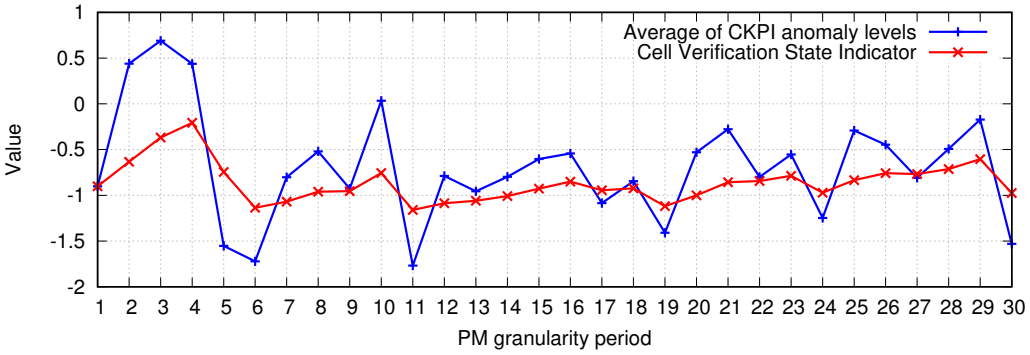


Figure 6.2: Exemplary computation of the CVSI. The state update factor α is selected as follows: 0.2 if $|\psi(\mathbf{a}^{\perp})| \in [0;1)$, 0.4 if $|\psi(\mathbf{a}^{\perp})| \in [1;2)$, and 0.8 if $|\psi(\mathbf{a}^{\perp})| \in [2;\infty)$.

cell. For example, function ψ can be the arithmetic average of $a_1^{\perp}, \ldots, a_n^{\perp}$ or the norm of vector $\mathbf{a}^{\perp}$ [CLRS09] .

It should be noted here that we may actually have more than one state update factor. The higher factor α is, the more impact would the observation at time t have on $\vartheta(\mathbf{a}^{\perp}, \alpha)_t$. However, there can be cases where we would like to update $\vartheta(\mathbf{a}^{\perp}, \alpha)_t$ differently. For instance, if its value is indicating a severe degradation, we may select a high α whereas in the case where a slight degradation is spotted we may choose to pick a low α. Such a technique is used in self-regulating algorithms like the one presented in [LPCS04].

Figure 6.2 represents an exemplary computation of $\vartheta(\mathbf{a}^{\perp}, \alpha)$ that follows that strategy. It makes use of three different update factors that depend on the current observation $\psi(\mathbf{a}^{\perp})$. The observation itself is computed as the arithmetic average of the elements of $\mathbf{a}^{\perp} = (a_1^{\perp}, \ldots, a_n^{\perp})$. As shown, over the 30 PM granularity periods the PM fluctuations have been smoothed which gives us a more realistic view on how a cell performs.

Finally, at time t a decision is made whether to further process a verification area. It is based on the values of $\vartheta(\mathbf{a}^{\perp}, \alpha)_t$ of all cells within an area.

6.3 Cell Clustering

Let us go one step further and expand the example from the previous section. Instead of having one cell, assume that we have a mobile network consisting of 12 cells and 18 cell adjacencies, as depicted in Figure 6.3. Now, suppose that the following changes are detected after accessing the CM database: CM change at cells 1, 2, 6, 8, and 11. Also, assume that cells 3, 4, 7, and 10 are marked as degraded after observing the PM database. Based on those events, let us construct a verification area by taking the target cell as well as its direct neighbors.

Immediately, those events lead to verification collisions. As shown in Figure 6.4(a), we have four verification collision pairs: $(1,2)$, $(2,6)$, $(6,8)$, and $(8,11)$. Here, the verification areas are identified by the ID of the target (reconfigured) cell. As we know

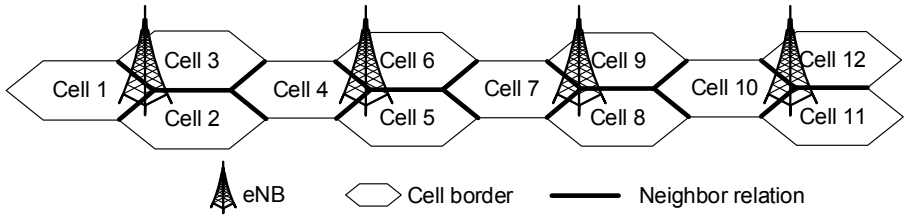


Figure 6.3: Example of a mobile network

from Definition 3.3, such collisions are an indication for uncertainties and prevent the simultaneous execution of corrective undo actions. A collision can be also seen as a safety mechanism that prevents the rollback of configuration changes that do not harm performance. However, collisions may also hold back actions without any reason if they are not justified. They are referred to as weak collisions, as extensively described in Section 3.2.4. In this section, the approach used to identify and eliminate such weak verification collisions is described.

In the terms of the SON verification concept, collisions occur when verification areas share anomalous cells (cf. Section 5.5). A verification area represents a set of cells that are grouped due to CM changes, in particular, the reconfigured cell and those potentially impacted by the change. However, the process of eliminating collisions leads to the exclusion of cells from a verification area, which has to be done with caution as it may induce unnecessary undo operations.

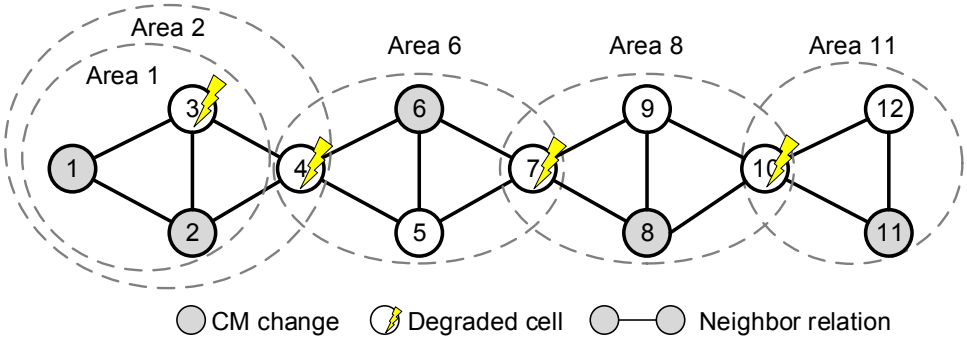
The starting point of eliminating a collision is the behavior of the cells. As outlined in Section 6.1, the cell behavior model is presented as a vector $\mathbf{a}^{\perp} = (a_1^{\perp}, \ldots, a_n^{\perp})$. To be able to exclude cells from an area, though, we need to group those that behave similarly. For this purpose, function d is defined, as given in Equation 6.6. It calculates the distance between two anomaly level vectors $\mathbf{a}_i^{\perp} \in A^{\perp}$ and $\mathbf{a}_j^{\perp} \in A^{\perp}$. A popular example is the Euclidean or the Manhattan distance function [DD09].

$$d \colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \tag{6.6}$$

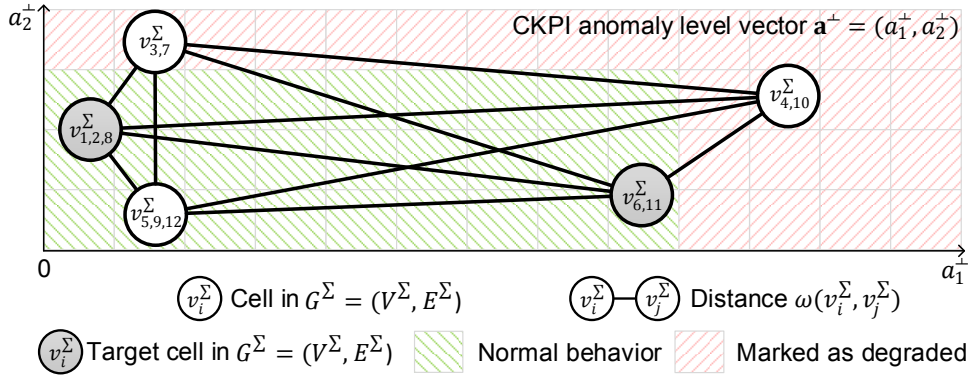
As a next step, we need to represent the cells as well as how they perform with respect to each other. For this purpose, an undirected edge-weighted *cell behavior graph* $G^{\Sigma} = (V^{\Sigma}, E^{\Sigma})$ is formed. Formally, it is specified as follows:

Definition 6.2 (Cell behavior graph)**.** A cell behavior graph $G^{\Sigma} = (V^{\Sigma}, E^{\Sigma})$ comprises of a set of vertexes V^{Σ} and a set of edges $V^{\Sigma} \times V^{\Sigma}$, denoted as E^{Σ}. Every vertex $v_i^{\Sigma} \in V^{\Sigma}$ represents a cell $\sigma_i \in \Sigma$ in the $\mathbb{R}^n$ space by taking its anomaly vector $\mathbf{a}_i^{\perp}$ into account. The weight of every edge $(v_i^{\Sigma}, v_j^{\Sigma}) \in E^{\Sigma}$, denoted as $w(v_i^{\Sigma}, v_j^{\Sigma})$, is computed by applying distance function d for $\mathbf{a}_i^{\perp} \in A^{\perp}$ and $\mathbf{a}_j^{\perp} \in A^{\perp}$.

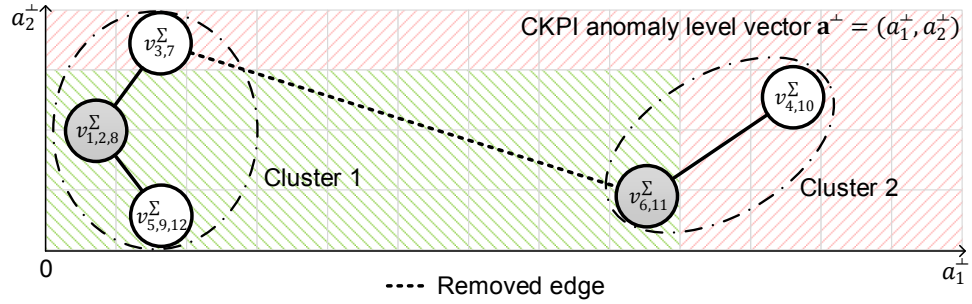
An exemplary composition of G^{Σ} in the $\mathbb{R}^2$ space is given in Figure 6.4(b). It should be noted that for simplicity reasons, some vertexes are completely overlapping, that is,



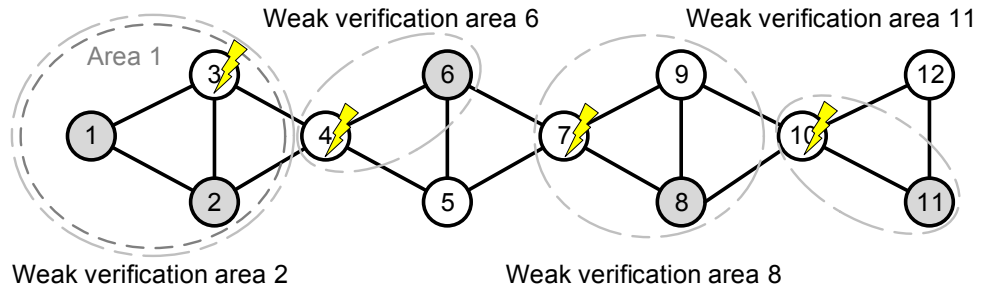
(a) Verification area formation by taking the first degree neighbors of the target cell. The identifier of a verification area equals the identifier of its target cell.



(b) Formation of the cell behavior graph $G^{\Sigma} = (V^{\Sigma}, E^{\Sigma})$ in $\mathbb{R}^2$. Two vertexes $v_i^{\Sigma}, v_j^{\Sigma} \in V^{\Sigma}$ are overlapping when cell i and j are represented by the same KPI anomaly level vector $\mathbf{a}^{\perp}$.



(c) Transformation of G^{Σ} into an MST. Cell clusters formed after the removal of the edge between nodes $v^{\Sigma}_{3,7}$ and $v^{\Sigma}_{6,11}$.



(d) Formation of weak verification areas. Cells are excluded from the initial verification area selection based on the cluster they have been assigned to.

Figure 6.4: Example of applying the MST-based clustering algorithm. It eliminates weak verification collisions by transforming the initially selected verification areas. The transformed areas are referred to as weak verification areas.

they represent cells showing exactly the same behavior. The following sets of vertexes represent cells fulfilling this criteria: $\{v_1^\Sigma, v_2^\Sigma, v_8^\Sigma\}$, $\{v_3^\Sigma, v_7^\Sigma\}$, $\{v_5^\Sigma, v_9^\Sigma, v_{12}^\Sigma\}$, $\{v_6^\Sigma, v_{11}^\Sigma\}$, and $\{v_4^\Sigma, v_{10}^\Sigma\}$. Moreover, each such set is represented by a single vertex $v_{\langle index \rangle}^\Sigma$ whose index is comprised of the indexes of the overlapping nodes. The distance between those nodes is zero which means that some edges are not depicted in the figure.

The next step towards eliminating weak verification collisions is to group the vertexes that represent similarly behaving cells. This procedure consists of two phases: (1) the formation of a Minimum Spanning Tree (MST), and (2) the removal of edges of that tree which leads to the formation of a forest of trees, each representing cells that behave alike. Algorithm 1 shows this procedure in detail.

The algorithm itself requires $G^\Sigma = (V^\Sigma, E^\Sigma)$ as input. Out of this graph an MST is generated, i.e., a connected, undirected, and acyclic graph $T^\Sigma = (\tilde{V}^\Sigma, \tilde{E}^\Sigma)$, where $\tilde{E}^\Sigma \subseteq E^\Sigma$ and $\tilde{V}^\Sigma = V^\Sigma$. Furthermore, the resulting tree minimizes $\sum_{v_i^\Sigma, v_j^\Sigma \in G^\Sigma} w(v_i^\Sigma, v_j^\Sigma)$. In order to form the MST, Kruskal's algorithm is used, which is characterized as being a greedy algorithm [Kru56, CLRS09]. The algorithm steps are given between lines 1 and 12. It starts by initializing the vertex and edge set of T^Σ, as well as a temporal list of edges that contains all elements of E^Σ. The latter one is sorted into increasing order (line 4) and is processed by continuously taking edges out of it (line 5). An edge and its adjacent vertexes are added to the tree only if they do not induce a loop (lines 6-10). The MST itself is formed in line 12.

During the second phase (lines 13-16), the vertexes of the tree $T^\Sigma = (\tilde{V}^\Sigma, \tilde{E}^\Sigma)$ are split

Algorithm 1: MST-based clustering algorithm

Input: Undirected, edge weighted cell behavior graph $G^\Sigma = (V^\Sigma, E^\Sigma)$
Result: Forest $F^\Sigma = \{\hat{T}_1^\Sigma, \ldots, \hat{T}_\kappa^\Sigma\}$

1 $\tilde{E}^\Sigma = \emptyset$;
2 $\tilde{V}^\Sigma = \emptyset$;
3 $List_{E^\Sigma} \leftarrow$ get elements of E^Σ;
4 sort $List_{E^\Sigma}$ into increasing order by weight w;
5 **foreach** *edge* $(v_1, v_2) \in List_{E^\Sigma}$ **do**
6 **if** $v_1 \notin \tilde{V}^\Sigma$ *and* $v_2 \notin \tilde{V}^\Sigma$ **then**
7 $\tilde{E}^\Sigma \leftarrow \tilde{E}^\Sigma \cup \{(v_1, v_2)\}$;
8 $\tilde{V}^\Sigma \leftarrow \tilde{V}^\Sigma \cup \{v_1\}$;
9 $\tilde{V}^\Sigma \leftarrow \tilde{V}^\Sigma \cup \{v_2\}$;
10 **end**
11 **end**
12 $T^\Sigma = (\tilde{V}^\Sigma, \tilde{E}^\Sigma)$;
13 $E_{temp} \leftarrow \xi(T^\Sigma)$;
14 $\tilde{E}^\Sigma \leftarrow \tilde{E}^\Sigma \setminus E_{temp}$;
15 find connected subgraphs $\{\hat{T}_1^\Sigma, \ldots, \hat{T}_\kappa^\Sigma\}$ in T^Σ;
16 $F^\Sigma = \{\hat{T}_1^\Sigma, \ldots, \hat{T}_\kappa^\Sigma\}$;

into κ groups. Furthermore, the minimum distance between vertexes of different groups is maximized. It is achieved by forming a forest F^{Σ}, i.e., an undirected graph whose connected components are trees, denoted as $\{\hat{T}_1^{\Sigma}, \ldots, \hat{T}_{\kappa}^{\Sigma}\}$, each being disjoint with the other. This procedure, also known as MST clustering [JN09], is carried out by removing a certain number of edges from T^{Σ} by starting from the longest one. For example, the removal of the two longest edges results into a forest of three trees, i.e., a group of three clusters.

The decision itself which one to remove is computed by an edge removal function ξ, as defined in Equation 6.7. The set of edges that are returned by ξ are taken out from the tree. One way of carrying out this operation is to remove edges that exceed a predefined threshold. Another way could be to use minimum entropy as criterion for clustering.

$$\xi \colon T^{\Sigma} \rightarrow \mathbb{P}(\tilde{E}^{\Sigma}) \tag{6.7}$$

Finally, the resulting forest $F^{\Sigma} = \{\hat{T}_1^{\Sigma}, \ldots, \hat{T}_{\kappa}^{\Sigma}\}$ is used to exclude cells from verification areas. Depending on how many edges were removed, one of the following three cases may occur:

- $\kappa = 1$
- $\kappa = |V^{\Sigma}|$
- $\kappa = [2, |V^{\Sigma}|)$

In the first case, the forest F^{Σ} has only one tree, i.e., we have only one cluster to which all cells have been assigned to. Consequently, they are seen as behaving in the same manner which impedes the removal of any collision. In the second case, the forest F^{Σ} has $|V^{\Sigma}|$ trees, which means that each tree consists of exactly one vertex and that no two cells are assigned to the same cluster. As a result, all verification collisions are marked as weak and are removed. In the third case, the number of clusters varies between two and $|V^{\Sigma}|$. Here, cells from a target extension set that are not assigned to the cluster of the target cell are excluded from the given verification area. Such areas are also referred to as *weak verification areas*.

Definition 6.3 (Weak verification area)**.** A verification area is called weak when at least one of its cells gets excluded from the initial area selection after the completion of the cell clustering process.

On the contrary, areas whose cells are all located within the same cluster remain unchanged. The verification collisions that have disappeared after this step are marked as weak and are eliminated.

Figures 6.4(c) and 6.4(d) visualize the continuation of the aforementioned example. The first one depicts an exemplary MST that may result after triggering the algorithm. It consists of the initial five vertexes and the following four edges: $(v^{\Sigma}_{\langle 1,2,8\rangle}, v^{\Sigma}_{\langle 5,9,12\rangle})$,

$(v^{\Sigma}_{\langle 1,2,8\rangle}, v^{\Sigma}_{\langle 3,7\rangle})$, $(v^{\Sigma}_{\langle 6,11\rangle}, v^{\Sigma}_{\langle 4,10\rangle})$, and $(v^{\Sigma}_{\langle 3,7\rangle}, v^{\Sigma}_{\langle 6,11\rangle})$. In addition, the latter edge is removed which leads to the formation of two clusters: $\{v^{\Sigma}_{\langle 1,2,8\rangle}, v^{\Sigma}_{\langle 5,9,12\rangle}, v^{\Sigma}_{\langle 3,7\rangle}\}$ and $\{v^{\Sigma}_{\langle 4,10\rangle}, v^{\Sigma}_{\langle 6,11\rangle}\}$.

The second figure gives us the end result. Four of the verification areas are converted to weak ones: area 2, 6, 8, and 11. Verification area 1, on the contrary, remains as initially selected. Hence, only the collision between area 1 and 2 is left over, while the remaining ones are removed.

6.4 Determining The Verification Collision Grade

In the previous section, an approach that simplifies the verification collision problem was introduced. It is based on an MST clustering technique that reduces the size of a verification area in order to eliminate weak collisions. In this section, though, the main focus is to evaluate the remaining collisions and define a term that represents the severity of the problem that needs to be solved.

First and foremost, it should be mentioned that we are no longer interested in cells individually, but in verification areas. As introduced in the previous section, some of those areas can be converted to weak ones after triggering the clustering approach, which means that some cells may have been excluded from the initial area selection. Therefore, also to simplify the notation, the set of all verification areas is denoted as Φ, whereas a single area as $\varphi \in \Phi$. Note that the set includes weak ones as well as those that remained unchanged.

Nevertheless, even after converting some areas to weak ones we may still have collisions present. Hence, let us form a so-called *verification collision graph* which is defined in the following way:

Definition 6.4 (Verification collision graph)**.** A verification collision graph $G^{\Phi} = (V^{\Phi}, E^{\Phi})$ is defined by a set V^{Φ} of vertexes that represent abnormally performing verification areas, and a set E^{Φ} of edges, each representing a verification collision $V^{\Phi} \times V^{\Phi}$. That is, two vertexes in G^{Φ} are connected with each other if and only if the corresponding undo actions must not be simultaneously executed.

As we can see, this graph is a representation of the verification collision problem which consequently means that we can apply techniques from graph theory to model the problem as well as to find the end result, i.e., generate a corrective action plan and solve any issues related to it.

The first technique that is used is minimum vertex coloring [BM82], which in its simplest form assigns each vertex a color such that no edge connects two vertexes having the same color. The vertexes that receive the same color can be seen as a group of objects that are fulfilling a certain criteria. This criteria itself is defined by the edges in the graph on which the coloring algorithm is applied.

Here, we have a verification collision graph G^{Φ} which, after coloring its vertexes, yields sets of nodes that represent independent sets of verification areas. Hence, the

undo actions associated with those areas can be executed at the same time. Formally, to perform minimum vertex coloring we need function m, as given in Equation 6.8. It assigns each node a positive natural number that represents a color. Note that the set of available colors is denoted as $\mathcal{C} \subset \mathbb{N}_0$ and that $|\mathcal{C}| = |V^\Phi|$, i.e., it is possible to assign each node with a different color.

$$m : V^\Phi \rightarrow \mathcal{C} \tag{6.8}$$

After assigning each vertex in G^Φ a color, we can determine the *verification collision grade*.

Definition 6.5 (Verification collision grade)**.** The smallest number of colors required to color G^Φ, also known as the chromatic number $\chi(G^\Phi)$, equals the verification collision grade of the verification collision problem.

The value of $\chi(G^\Phi)$ shows the number of sets of collision free CM undo operations. Thereby, it also represents the number of correction window slots (cf. Section 5.6) that would be required to process all actions. Furthermore, based on $\chi(G^\Phi)$ we can also determine whether G^Φ is *collision-complete* or *collision-free*.

Definition 6.6 (Collision-complete and collision-free verification collision graph)**.** A verification collision graph $G^\Phi = (V^\Phi, E^\Phi)$ is called to be *collision-complete* if and only if $m(v_i^\Phi) \neq m(v_j^\Phi)$ and *collision-free* if $m(v_i^\Phi) = m(v_j^\Phi)$ for all adjacent v_i^Φ and v_j^Φ in V^Φ.

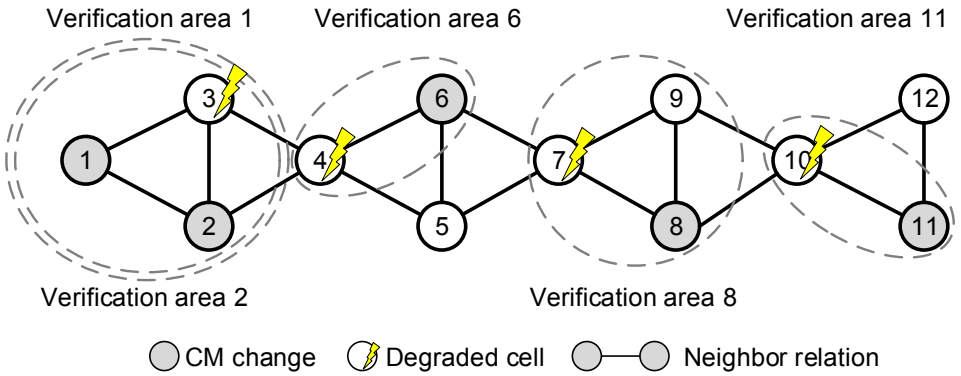
However, being collision-free or collision-complete is only an edge case property of the collision graph. In total, we have to distinguish between the following three outcomes:

- $\chi(G^\Phi) = 1$
- $\chi(G^\Phi) = |\mathcal{C}|$
- $\chi(G^\Phi) \in [2; |\mathcal{C}|)$ for $|V^\Phi| > 2$

The first two represent the collision-free and collision-complete property, respectively. Hence, all undo actions will either be allocated to the same time slot, or each will be assigned to a different one. In the third case, a portion of undo actions will be added to the same slot.

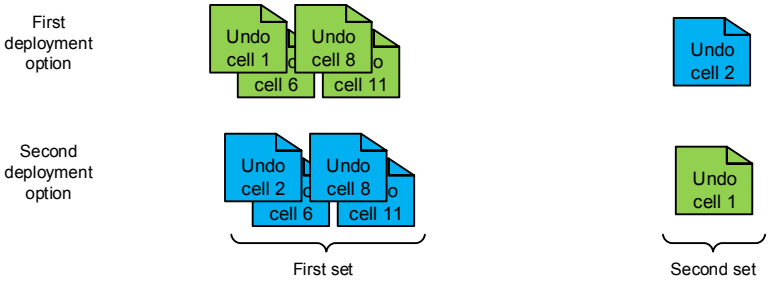
In addition, the chromatic number $\chi(G^\Phi)$ is also an indication whether we are dealing with an over-constrained corrective action plan (cf. Definition 3.7). Should, for instance, $\chi(G^\Phi)$ exceed the number of available correction window slots, it may not be possible to process all undo actions in time. This is also the reason why minimum vertex coloring is not sufficient to solve the verification collision problem.

Figure 6.5 shows an example representing the technique that has been discussed so far. It is a continuation of the example presented in Figure 6.4, which originates from the exemplary network of 12 cells and 18 cell adjacencies, as given in Figure 6.3.



(a) Verification area formation as computed by the MST-based clustering algorithm. The identifier of a verification area equals the identifier of its target cell.

(b) Estimating the severity of the verification collision problem. The verification collision grade is computed by applying minimum vertex coloring on the verification collision graph G^{Φ}. Here, it results in $\chi(G^{\Phi}) = 2$.



(c) The verification collision grade $\chi(G^{\Phi}) = 2$ results in two permissible undo action deployments. Each set consists of undo actions that are not in conflict with each other.

Figure 6.5: Example of estimating the verification collision grade of the verification collision graph $G^{\Phi} = (V^{\Phi}, E^{\Phi})$

In total, we have five verification areas and one collision between area 1 and 2, as Figure 6.5(a) outlines. Note that the area identifier equals the ID of the given target cell. As a result, we get a graph G^{Φ} that consists of the following set of vertexes and edges: $V^{\Phi} = \{v_1^{\Phi}, v_2^{\Phi}, v_6^{\Phi}, v_8^{\Phi}, v_{11}^{\Phi}\}$ and $E^{\Phi} = \{(v_1^{\Phi}, v_2^{\Phi})\}$.

Figure 6.5(b) visualizes the result after applying minimum vertex coloring. The verification collision grade equals to 2 which means that we would need two correction window time slots to process all undo actions. Furthermore, there are two ways of coloring G^{Φ} since v_6^{Φ}, v_8^{Φ}, and v_{11}^{Φ} may receive either of the two colors. Hence, there are two possible ways of deploying the undo actions. Figure 6.5(c) depicts the two permissible undo action deployments.

6.5 Solving a Verification Collision Problem

Let us continue with the example from Figure 6.5. It represents the verification areas that have been formed around the reconfigured cells as well as the resulting verification collision graph $G^{\Phi} = (V^{\Phi}, E^{\Phi})$, as given by Definition 6.4. As it can be noticed, the undo actions for cell 1 and 2 cannot be executed at the same time, hence, we need to find out which of those two requests are most likely responsible for the degradation of cell 3.

Finding a solution for this problem means that we need to solve a constraint satisfaction problem [RvBW06]. In its general form, it is characterized by a set of constraints, variables, and values for those variables [RN10]. In terms of the verification collision problem, the variables represent verification areas, the values the correction window slots, whereas the constraints the collisions. In this section, each of those sets is going to be described.

First of all, let us define a bijective assignment function, as given in Equation 6.9, that maps each vertex $v_i^{\Phi} \in V^{\Phi}$ to a *variable* $x_i \in X$, where X is the set of all variables and $|X| = |V^{\Phi}|$. Furthermore, let each x_i be in $\mathbb{N}^+$ and let it take values between 1 and τ, i.e., the number of available correction window slots. This is equivalent to say that an undo action associated with a verification area is assigned to one of the slots.

$$\omega \colon V^{\Phi} \rightarrow X \tag{6.9}$$

As a next step, let the set of all *constraints* be denoted as Θ. In addition, let a constraint be a pair $(x_1, x_2) \in \Theta$ of variables $x_1, x_2 \in X$ that disallows their equality. As a consequence, we get the following interpretation:

$$\forall (v_1^{\Phi}, v_2^{\Phi}) \in E^{\Phi} : x_1 \neq x_2, \tag{6.10}$$

where $\omega(v_1^{\Phi}) = x_1$ and $\omega(v_2^{\Phi}) = x_2$. The variables associated with vertexes connected in G^{Φ} must not receive the same value, i.e., the set of edges E^{Φ} can be also seen as a set of constraints. It should be noted here that in this thesis they are referred to as *hard constraints*.

Finally, the objective function is specified. However, before doing so we need to introduce priority function ρ, as shown in Equation 6.11. The outcome should be interpreted as follows: the lower the value of x_i, the more important it is to process the undo action for the area associated with v_i^{Φ}. The priority itself may depend on factors like the geographical location of the target cell, the number of degraded cells within a verification area or the overall degradation level.

$$\rho \colon X \rightarrow \mathbb{R} \tag{6.11}$$

The undo action deployment is then defined as a constraint optimization problem, as

follows:

$$\max \sum_{x_i \in X} \rho(x_i) x_i \tag{6.12}$$

subject to

$$\forall (x_i, x_j) \in \Theta \tag{6.13}$$

The objective is given in Equation 6.12, which is a maximization of the sum of the variables multiplied by the assigned priorities. Thereby, the lower $\rho(x_i)$ is, the lower the value of the corresponding variable x_i will be. In addition, the undo actions for the areas associated with the variable receiving the value of 1 are allocated to the first correction window slot and are, therefore, executed at first place.

Finally, the impact of those undo actions on the network performance is assessed. Should the network still show an anomalous behavior, the whole process is triggered again. However, with one notable difference, namely a decreased τ. The reason is that we have already consumed one time slot after deploying the first set of undo requests, i.e., we have one less for completing the CM verification process.

Figure 6.5 visualizes an exemplary outcome of this process. It shows the verification collision graph $G^\Phi = (V^\Phi, E^\Phi)$ as well as the outcome of the variable assignment step. In particular, we have the following variables: x_1, x_2, x_6, x_8, and x_{11}. In addition, there is only one constraint, namely $x_1 \neq x_2$ which results from the edge $(v_1^\Phi, v_2^\Phi) \in E^\Phi$. Each variable x_i is multiplied with the outcome of $\rho(x_i)$ and the resulting weighted sum is maximized. As shown, one permissible outcome is all variables except x_2 to receive value 1. Consequently, the undo actions for cells 1, 6, 8 and 11 are allocated to the first correction window time slot, as Figure 6.7 outlines.

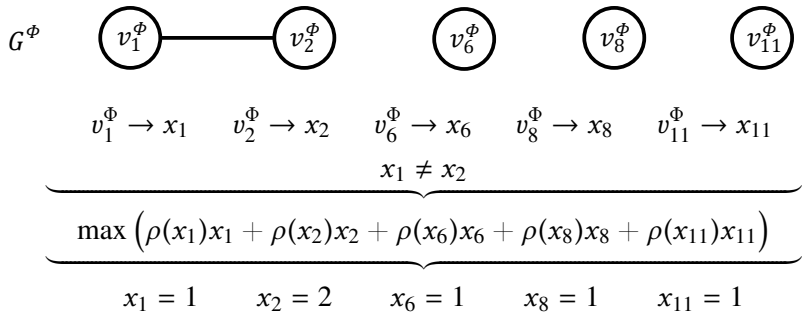


Figure 6.6: Variable assignment and constraint definition example. A node v_i^Φ in G^Φ is represented by a variable $x_i \in [1; \tau]$, where $\tau = 2$. An edge (v_i^Φ, v_j^Φ) adds the constraint $x_i \neq x_j$.

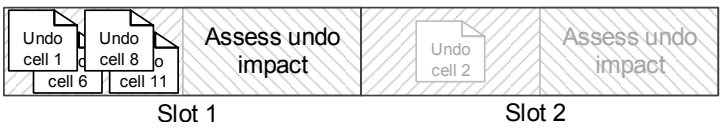


Figure 6.7: Correction window slot allocation. The order is defined by the variable assignment process from the previous step.

6.6 Solving an Over-Constrained Verification Collision Problem

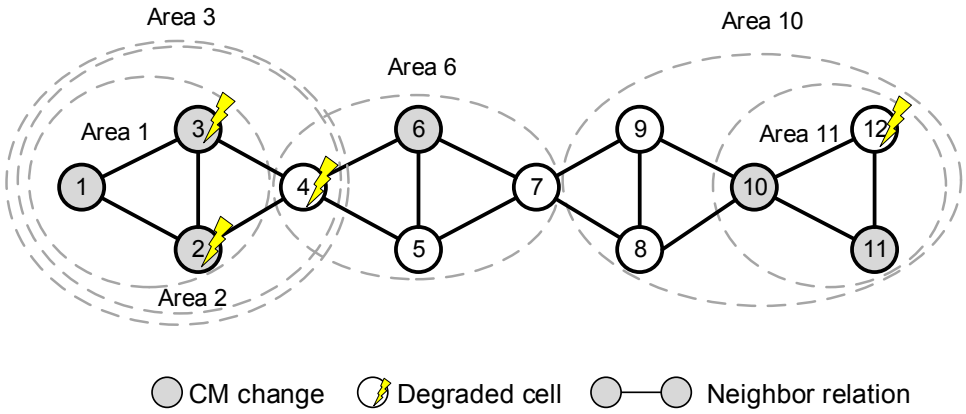
Sections 6.1 to 6.5 presented the cell behavior model, described how to prevent verification areas from being unnecessarily processed, how to eliminate weak collisions as well as how to resolve those being considered as valid. However, providing a solution to the verification collision problem might be a challenging task, as shown in the following example. Suppose that our network, introduced in the very beginning of this chapter (cf. Figure 6.3), shows an activity like the one outlined in Figure 6.8(a). Namely, cells 1, 2, 3, 6, 10 and 11 have been reconfigured and cells 2, 3, 4, and 12 have degraded. Consequently, we have collisions between the following pairs of verification areas: (1,2), (1,3), (2,3), (2,6), (3,6), and (10,11). Note that the identifier of an area equals the identifier of the corresponding target cell. Now, assume that all collisions are valid, i.e., no collision was eliminated after the clustering process, as described in Section 6.3. In addition, let the number of available correction window slots τ be two. Obviously, we have an over-constrained corrective action plan, as stated by Definition 3.7, since we cannot find a slot allocation that satisfies all constraints. We can visualize that by applying minimum vertex coloring of the resulting verification collision graph $G^{\Phi} = (V^{\Phi}, E^{\Phi})$, as depicted in Figure 6.8(b). In total, three colors are required, that is, a verification collision grade $\chi(G^{\Phi})$ of 3 (cf. Definition 6.5). Hence, we would need three time slots to process all undo actions.

As a result, we cannot use the verification collision resolving approach that has been introduced in Section 6.5. Instead, it has to be modified in such a way that an *acceptable solution* can be found by identifying the set of soft verification collisions (cf. Definition 3.8). However, softening some collisions means that valid constraints are going to be removed in order to put the corrective action plan $\mathcal{P}^{C^{\perp}} = \{\hat{C}_1^{\perp}, \ldots, \hat{C}_i^{\perp}\}$ (cf. Definition 3.4) within the expected limits, i.e., $|\mathcal{P}^{C^{\perp}}| \leq \tau$. Hence, we have a constraint optimization problem for which we need to minimize the total constraint violation.

First of all, to find such a solution we have to identify the vertexes in the verification collision graph $G^{\Phi} = (V^{\Phi}, E^{\Phi})$ that make the problem unsolvable. Cliques from graph theory [CLRS09] are able to provide that type of information. A clique $\bar{V}^{\Phi}$ is a subset of the vertexes of the graph G^{Φ} (i.e., $\bar{V}^{\Phi} \subseteq V^{\Phi}$) such that every two distinct vertexes $v_i^{\Phi}, v_j^{\Phi} \in \bar{V}^{\Phi}$ are adjacent in G^{Φ}, i.e., $(v_i^{\Phi}, v_j^{\Phi}) \in E^{\Phi}$. As a result, a clique induces a complete subgraph of G^{Φ} which consequently means that all areas associated with the vertexes within a clique are in collision with each other.

In addition, the focus is to find only those cliques that have a certain size, namely such that have more than τ vertexes upon all *maximal cliques*. As known from graph theory, a maximal clique is a clique that cannot be extended by adding more adjacent vertexes [CLRS09]. Consequently, each clique having more than τ vertexes represents an over-constrained verification collision problem (i.e., an over-constrained corrective action plan) as the collision grade of that subgraph would exceed the total number of available correction window slots τ.

Figure 6.9 visualizes the outcome of this particular step. The given verification colli-



(a) Verification area formation by taking the first degree neighbors of the target cell. The identifier of a verification area equals the identifier of its target cell.

(b) The resulting verification collision graph G^{Φ}. Each color identifies a set of collision free undo actions. The verification collision grade is 3, i.e., $\chi(G^{\Phi}) = 3$. Hence, it is impossible to create a corrective action plan the size of 2.

Figure 6.8: An example of an over-constrained verification collision problem

Figure 6.9: Clique search in the verification collision graph G^{Φ}. In total, there are two maximal cliques exceeding the limit $\tau = 2$: $\bar{V}_1^{\Phi}$ and $\bar{V}_2^{\Phi}$. Both are comprised of three vertexes.

sion graph $G^{\Phi} = (V^{\Phi}, E^{\Phi})$ is represented by the vertex set $V^{\Phi} = \{v_1^{\Phi}, v_2^{\Phi}, v_3^{\Phi}, v_6^{\Phi}, v_{10}^{\Phi}, v_{11}^{\Phi}\}$, and the edge set $E^{\Phi} = \{(v_1^{\Phi}, v_2^{\Phi}), (v_1^{\Phi}, v_3^{\Phi}), (v_2^{\Phi}, v_3^{\Phi}), (v_2^{\Phi}, v_6^{\Phi}), (v_3^{\Phi}, v_6^{\Phi}), (v_{10}^{\Phi}, v_{11}^{\Phi})\}$. Due to the limitation of $\tau = 2$, the maximal cliques that we are interested in must contain at least three vertexes. In this example, two are fulfilling this requirement: $\bar{V}_1^{\Phi} = \{v_1^{\Phi}, v_2^{\Phi}, v_3^{\Phi}\}$, and $\bar{V}_2^{\Phi} = \{v_2^{\Phi}, v_3^{\Phi}, v_6^{\Phi}\}$.

Now, we need to determine whether those vertexes are part of one single over-constrained problem, or whether they are part of two or more independent problems. Obviously, splitting the graph into cliques does not yield separate over-constrained problems since cliques may have common vertexes, e.g., as $\bar{V}_1^{\Phi}$ and $\bar{V}_2^{\Phi}$ do. As a result, we

need to find another way of splitting the set of all nodes being part of a clique. Let us denote the union of all found cliques as $\bigcup_{i=1}^{n} \bar{V}_i^{\Phi}$. Considering the above-mentioned example, we would get $\bigcup_{i=1}^{n} \bar{V}_i^{\Phi} = \{v_1^{\Phi}, v_2^{\Phi}, v_3^{\Phi}, v_6^{\Phi}\}$, where $n = 2$.

Formally, we must find a partition of $\bigcup_{i=1}^{n} \bar{V}_i^{\Phi}$, denoted as $\mathcal{P}(\bigcup_{i=1}^{n} \bar{V}_i^{\Phi})$, that yields sets of independent verification problems. If we unite cliques that are sharing vertexes and consider the resulting unions as one entity, we will get such a split. In this thesis, the union of such cliques, i.e., the block of the resulting partition, is called a *clique group*. Also, the properties of a partition are valid, i.e., $\mathcal{P}$ is a set of sets that does not contain the empty set, the union of all sets in $\mathcal{P}$ equals $\bigcup_{i=1}^{n} \bar{V}_i^{\Phi}$, and the intersection of any two sets in $\mathcal{P}$ is empty.

Definition 6.7 (Clique group)**.** A clique group is a block of a partition $\mathcal{P}(\bigcup_{i=1}^{n} \bar{V}_i^{\Phi})$, where $\bigcup_{i=1}^{n} \bar{V}_i^{\Phi}$ is the union of all found maximal cliques $\bar{V}_i^{\Phi}$ in the verification collision graph $G^{\Phi} = (V^{\Phi}, E^{\Phi})$. The partition is formed by uniting cliques $\bar{V}_i^{\Phi}$ that share common vertexes. Cliques that do not share vertexes form their own clique group.

In the above-mentioned example, we will get the partition $\mathcal{P}(\bigcup_{i=1}^{n} \bar{V}_i^{\Phi}) = \{\{v_1^{\Phi}, v_2^{\Phi}, v_3^{\Phi}, v_6^{\Phi}\}\}$, i.e., the partition is a singleton set since we have exactly one clique group that unites the elements of $\bar{V}_1^{\Phi}$ and $\bar{V}_2^{\Phi}$.

Each clique group represents an over-constrained problem for which an acceptable solution has to be found. The way of doing that is by removing edges from a clique group and merging the adjacent vertexes. This process is also known as edge contraction [GY05], i.e., for every two merged vertexes, an edge $e^{\Phi} \in E^{\Phi}$ is removed and its two incident vertexes $v_i^{\Phi}, v_j^{\Phi} \in V^{\Phi}$, are merged into a new vertex $\check{v}_k^{\Phi}$, where the edges incident to $\check{v}_k^{\Phi}$ each correspond to an edge incident to either v_i^{Φ} or v_j^{Φ}. This procedure gives us a new undirected graph $\check{G}^{\Phi} = (\check{V}^{\Phi}, \check{E}^{\Phi})$ that does not have the edges between merged vertexes. At the end, the number of vertexes within each clique group must not exceed τ.

In order to find those edges, each vertex $v_i^{\Phi} \in V^{\Phi}$, that is also being part of a clique group, is mapped to a variable $\tilde{x}_i \in \tilde{X}$ ranging between 1 and τ. Similarly to Section 6.5, we have a function $\tilde{\omega}\colon V^{\Phi} \to \tilde{X}$ that carries out this assignment. In addition, all edges $V^{\Phi} \times V^{\Phi}$ within a clique group are considered as constraints in the same way as given in Equation 6.10, i.e., an edge (v_i^{Φ}, v_j^{Φ}) leads to $(\tilde{x}_i, \tilde{x}_j)$, i.e., $\tilde{x}_i \neq \tilde{x}_j$, to be added to the set of constraints, denoted as $\tilde{\Theta}$. In contrast to Section 6.5, those constraints are referred to as *soft constraints*, which as known from constraint optimization, are those that can be violated depending on the priority they have received [RvBW06]. The priority itself is computed by function $\tilde{\rho}$, defined as follows:

$$\tilde{\rho}\colon \tilde{\Theta} \to \mathbb{R} \tag{6.14}$$

Compared to the approach introduced in the previous section, it rates a verification collision (edge in G^{Φ}) instead of a verification area (vertex in G^{Φ}). In addition, the outcome of $\tilde{\rho}$ should be interpreted as follows: the higher the value, the more important it is the constraint to be satisfied, i.e., the collision to remain.

Finally, the clique group size reduction is modeled as a constraint optimization problem defined in the following manner:

$$\max \sum_{(\tilde{x}_i, \tilde{x}_j) \in \tilde{\Theta}} \tilde{\rho}(\tilde{x}_i, \tilde{x}_j) r(\tilde{x}_i, \tilde{x}_j) \tag{6.15}$$

subject to

$$r(\tilde{x}_i, \tilde{x}_j) = \begin{cases} 0 & \text{iff } \tilde{x}_i = \tilde{x}_j \\ 1 & \text{otherwise} \end{cases} \tag{6.16}$$

As known from constraint optimization, Equation 6.16 gives us a reification function r that indicates whether the inequality of two variables $\tilde{x}_i$ and $\tilde{x}_j$ is satisfied. Equation 6.15 defines the objective, i.e., the maximization of the weighted sum of all soft constraints. At the end, the vertexes whose variables receive the same value are merged together.

Figure 6.10 visualizes an example of this procedure. Each of the four vertexes of the clique group in Figure 6.9 is mapped to a variable $\tilde{x}_i$. In addition, the edges within the group identify the constraints, i.e., $\tilde{\Theta}$ consist of $\tilde{x}_1 \neq \tilde{x}_2$, $\tilde{x}_1 \neq \tilde{x}_3$, $\tilde{x}_2 \neq \tilde{x}_3$, $\tilde{x}_2 \neq \tilde{x}_6$, and $\tilde{x}_3 \neq \tilde{x}_6$. One permissible outcome could be the merge of v_2^Φ, v_3^Φ, and v_6^Φ, since their variables got the same value. Hence, the edges (v_2^Φ, v_3^Φ), (v_2^Φ, v_6^Φ), (v_3^Φ, v_6^Φ) are removed from the graph. The resulting new graph $\check{G}^\Phi$ (cf. Figure 6.11(a)) consists of four vertexes $\check{v}_1^\Phi$, $\check{v}_{\langle 2,3,6 \rangle}^\Phi$, $\check{v}_{10}^\Phi$, and $\check{v}_{11}^\Phi$, as well as the edges $(\check{v}_1^\Phi, \check{v}_{\langle 2,3,6 \rangle}^\Phi)$ and $(\check{v}_{10}^\Phi, \check{v}_{11}^\Phi)$. Note that the merged vertexes are identified by $\check{v}_{\langle 2,3,6 \rangle}^\Phi$ in $\check{G}^\Phi$ whereas the remaining ones keep the initial identifiers.

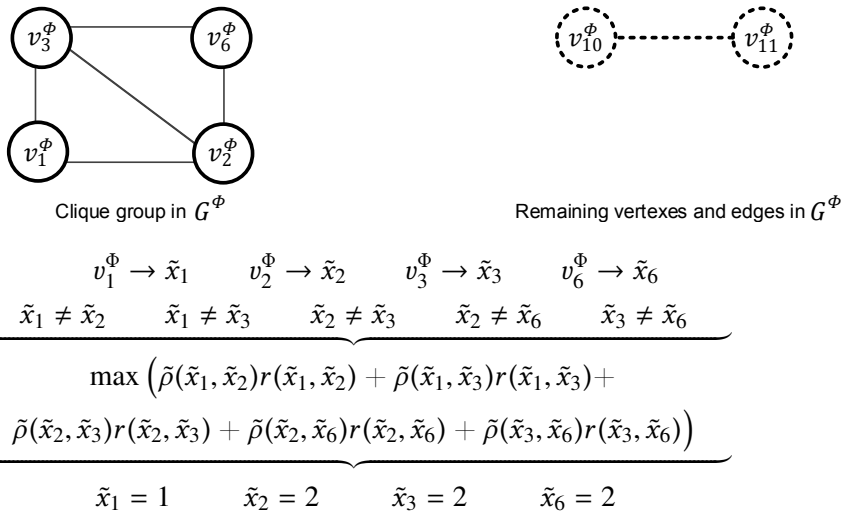


Figure 6.10: Solving an over-constrained verification collision problem. Each vertex v_i^Φ being part of a clique group is assigned to a variable $\tilde{x}_i \in [1; \tau]$, where $\tau = 2$. An edge (v_i^Φ, v_j^Φ) within a group represents a soft constraint $\tilde{x}_i \neq \tilde{x}_j$. The outcome of the maximization process determines the vertexes that are going to be united.

(a) Reducing the size of a clique group. The usage of soft constraints allows us to find an acceptable correction window slot allocation. After carrying out the constraint optimization process, we get a new graph $\check{G}^{\Phi}$ which is no longer comprised of over-sized cliques.

(b) Collision grade $\chi(\check{G}^{\Phi})$ of the reduced verification collision graph $\check{G}^{\Phi}$ is 2. Hence, it is possible to execute the actions in two iterations.

Figure 6.11: Outcome of the edge contraction procedure

After completing those steps, there is no longer an over-constrained verification collision problem, as the verification collision grade (cf. Definition 6.5) no longer exceeds the number of available correction window time slots. As shown in Figure 6.11(b), its value equals to 2. As a result, we can continue with the approach introduced in Section 6.5 which finally generates the corrective action plan (cf. Definition 3.4).

6.7 Summary

This chapter presented the part of the verification process that is responsible for the assessment of configuration change and the rollback of those harming network performance. It is modeled as a three step procedure that based on the ongoing CM changes generates verification areas, assesses them, and assembles a corrective action plan. Throughout this chapter, the cell behavior model, the strategy used for detecting anomalies, and the generation of the plan have been discussed. The latter one consists of multiple steps: clustering cells based on their behavior, estimating the severity of the verification collision problem, modeling it as a constraint optimization problem, and finally solving it in order to provide a set of undo actions. A strategy for solving an over-constrained optimization problem has been introduced as well.

The cell behavior model and the presented detection method (cf. Sections 6.1 and 6.2) contribute to the following three research objectives:

O2.2: Definition of the verification state.
Q: How to model the behavior of a cell?
A: The behavior of a cell is defined by its CKPI anomaly vector where each element of the vector is called a CKPI anomaly level (cf. Definition 6.1). The latter one shows the deviation of a CKPI from the expected value. The vector itself is computed by

taking the profile and the current CKPIs of a cell (cf. Equation 6.1). Furthermore, the collection of all CKPI anomaly level vectors is referred to as the CKPI anomaly level vector space.

O2.3: Estimate the duration of the verification process.
Q: How to handle fluctuations in the PM data and prevent function transactions from being interrupted?
A: The ability to handle fluctuations in the PM data is realized by computing a simple weighted average of the current observation, denoted as $\psi(\mathbf{a}^{\perp})_t$, and the previous smoothed $\vartheta(\mathbf{a}^{\perp}, \alpha)_{t-1}$, as given in Equation 6.4. The parameter that influences the outcome is the state update factor α, as used in the same equation. Its selection also affects the duration of the verification process since it can prevent verification areas from being further assessed.

Furthermore, in this chapter graph theory concepts have been utilized to model the verification collision problem, as introduced in Section 3.2.2. They contribute to the following objective:

O3.1: Define uncertainties in the terms of a verification process.
Q: How to model the verification collision problem?
A: The verification collision problem is modeled by using concepts and techniques known from graph theory. In particular, a verification collision graph (cf. Definition 6.4) is formed that shows which undo actions can and which must not be simultaneously executed. All further verification steps, e.g., the procedure that resolves collisions, use this graph as baseline.

The process of generating a corrective action plan (cf. Sections 6.3 to 6.6) utilizes concepts from graph theory and constraint optimization. In particular, a contribution to the following research objectives is made:

O3.3: Resolve and eliminate uncertainties and provide accurate corrective actions when verifying configuration changes.
Q: How to eliminate weak verification collisions?
A: Before starting the collision resolving procedure, those being identified as weak are eliminated. For this purpose, a cell behavior graph is formed (cf. Definition 6.2) which is used as input by the MST-based cell clustering algorithm. The clustering mechanism is responsible for the grouping of similarly behaving cells as well as the exclusion of cells from the initially defined verification areas. The exclusion itself leads to the formation of weak verification areas (cf. Definition 6.3) which enables the elimination of weak verification collisions.

Q: How to solve the verification collision problem and generate a corrective action plan?
A: The verification collision problem is modeled as a constraint optimization problem, as given in Equation 6.12. Concretely, each node in verification collision graph is represented by a variable that may accept values between 1 and the maximum number of available correction window time slots. The verification collisions define the so-called hard constraints which must never be violated. In addition, a priority function that indicates the importance of an undo action to be processed is defined. After carrying out the optimization, the assigned values to the variables determines the undo action execution order.

Q: How to detect an over-constrained verification collision problem?
A: An over-constrained verification collision problem is detected after applying minimum vertex coloring on the verification collision graph (cf. Definition 6.4). It is marked as being such if the resulting chromatic number, also called the verification collision grade (cf. Definition 6.5), exceeds the maximum number of correction window slots. The exact location of an over-constrained verification collision problem is determined after the search for maximal cliques in the verification collision graph. This process leads to the formation of clique groups (cf. Definition 6.7), each representing a separate over-constrained verification problem.

Q: How to model soft verification collisions and how to find an appropriate corrective action plan?
A: As discussed in Section 6.6, soft verification collisions are modeled as soft constraints in a constraint optimization problem (cf. Equation 6.15). It minimizes the total constraint violation by assigning those undo actions to the same corrective action window slot that have the lowest probability of rolling back unharmful changes.

It should be noted that the process of eliminating verification collisions (uncertainties), in particular, the formation of weak verification areas, also contributes to the research objective that targets the verification scope definition:

O2.1: Fragmentation of the network and specifying the scope of verification.
Q: Can the initially defined verification scope change when verifying CM changes?
A: Yes. During the process of eliminating weak verification collision (cf. Section 6.3) verification areas can be transformed into weak ones. That is, cells are taken out from an area in order to remove weak collisions (cf. Definition 3.9). The decision is based on the outcome of the MST-based clustering algorithm.

The presented approach also estimates the severity as well as defines several properties that further evaluate the verification collision problem. It contributes to the following research objective:

O3.4: Estimate the severity of the CM verification problem.
Q: How to estimate the severity of the verification collision problem?
A: The severity of the CM verification problem depends on the severity of the verification collision problem. The latter one is reflected by the chromatic number $\chi(G^{\Phi})$, also called the verification collision grade (cf. Definition 6.5), after applying minimum vertex coloring on the verification collision graph (cf. Definition 6.4). It shows the maximum number of correction window slots that are required to deploy the necessary undo actions and resolve all verification collisions.

Q: When is the verification problem collision-complete and when collision-free?
A: The problem is called collision-complete in case the verification collision grade $\chi(G^{\Phi})$ equals the number of formed verification areas and collision-free when the grade equals to 1 (cf. Definition 6.6). Collision-complete also means that every two undo actions must not be simultaneously deployed whereas free that all undo actions are allowed to be executed at the same time, i.e., allocated to the first time slot of the correction window (cf. Section 5.6).

Q: Why is minimum vertex coloring insufficient to solve the verification collision problem?
A: Although minimum vertex coloring is used to estimate the verification collision grade, it cannot not be used for resolving collisions and generating a corrective action plan. It does not specify the action execution order and cannot identify soft verification collisions (cf. Definition 3.8).

Chapter 7

Verification of Topology Changes

The general question that is going to be answered in this chapter is how to overcome verification-related issues that emerge due to dynamic topology changes. Section 3.2.5 already gave a comprehensive overview of those problems. In short, switching cells on or off may induce uncertainties while verifying configuration changes. Dynamic topology changes may result in incomplete profiles, anomalous cell behavior, and the rollback of necessary configuration changes. In addition, anomalies induced by the movement of UEs cannot be eliminated by solely using the CM verification process (cf. Section 5.1). In the presence of topology changes it may generate a weak corrective action plan (cf. Definition 3.13), i.e., the process is not always capable of providing a corrective action. This limitation comes from the fact that the up to now introduced verification process is not allowed to do more than rolling back already deployed CM changes. As a result, we need a verification strategy that actively monitors and adapts the network topology.

Generally speaking, this issue can be represented as a Steiner tree problem [GHNP01] which is very similar to the MST problem [CLRS09]. For a given undirected edge weighted graph, we have to find a tree that interconnects all vertexes and, at the same time, is of shortest length. The difference between the MST and the Steiner tree problem is that to solve the latter one we may include *extra* vertexes to the graph in order to reduce the length of the spanning tree. In literature, those new vertexes are called *Steiner points*, whereas the initial nodes of the graph are referred to as *Steiner terminals*, or fixed terminals.

Nevertheless, the Steiner tree algorithm itself cannot be applied in terms of the verification context without defining all of its properties. First of all, we need to map the existing network topology to a graph which serves as input to the algorithm, i.e., we must find an appropriate representation of the deployed cells and neighbor relations. Second, we have to represent the issue of verifying topology changes as an optimization problem by introducing a metric that requires minimization. This requirement comes from the fact that the Steiner algorithm minimizes the total weight of the spanning tree based on the edge weights in the input graph. Third, we need to specify which entities are actually

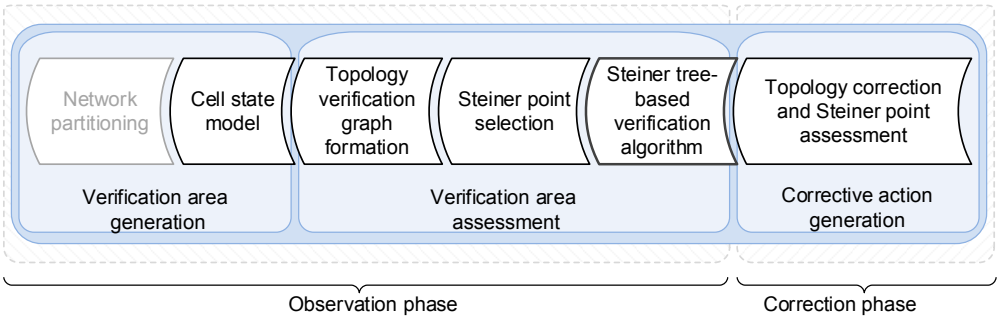


Figure 7.1: Overview of the topology verification process

marked as terminals and which as Steiner points. Defining an entity as a Steiner point does not necessarily mean that it is going to be selected to form the tree. Moreover, there are cases where such a point is never selected by the algorithm, e.g., if is a leaf in the input graph.

Throughout this chapter, those challenges are discussed and addressed by the so-called *topology verification process* which is the second main building block of the SON verification concept (cf. Section 5.1). Figure 7.1 shows a high-level overview of this process. At first, we have the cell state model which addresses the issue of how to model a cell's behavior. Only after the specification of the cell model we can use the Steiner tree algorithm. Section 7.1 discusses this topic as well as outlines the differences to the model utilized by CM verification. In addition, it discusses how to overcome the issue of having incomplete profiles. Second, in Section 7.2 the formation of the topology verification graph, i.e., the input graph of the Steiner tree algorithm, is described. Third, in Section 7.3 the algorithm is presented in detail. Fourth, Section 7.4 is devoted to the generation of corrective topology actions based on the algorithm's outcome.

In addition, in Section 7.5 the limitations of the Steiner tree-based verification algorithm are analyzed. It discusses the impact of the verification area selection on the algorithm as well as the edge cases that may emerge. Finally, Section 7.6 summarizes all sections and lists the answers to the research questions that have been given throughout the chapter.

Published work This chapter is based on the work made in [TATSC16b]. The initial idea of having a topology verification process is outlined in [TATSC16c]. Compared to those papers, this chapter goes in more detail. In addition, the notation has been adapted in order to present the concept in a uniform way.

7.1 Cell State Model

In terms of the topology verification problem, there are two cell types: *static* and *on-demand cells*. Within the first class fall cells that are never turned off whereas the second one represents cells that can be enabled or disabled during their operation. Let the set of all static cells be denoted as $\dot{\Sigma}$ whereas the set of all on-demand cells as $\mathring{\Sigma}$. Furthermore, let the set of all cells be denoted as Σ as well as let $\dot{\Sigma} \cup \mathring{\Sigma} = \Sigma$ and $\dot{\Sigma} \cap \mathring{\Sigma} = \emptyset$.

Similarly to the cell behavior model of the CM verification process (cf. Section 6.1), those cells are represented by the anomaly levels of the selected KPIs. It should be noted that they are referred to as TKPIs (cf. Definition 5.6). For each TKPI $k^{\vdash}$, a *TKPI anomaly level* is computed. It is denoted as $a^{\vdash}$ and is characterized in the following way:

Definition 7.1 (TKPI anomaly level)**.** A TKPI anomaly level $a^{\vdash}$ is an element of a TKPI anomaly level vector $\mathbf{a}^{\vdash} = (a_1^{\vdash}, \ldots, a_n^{\vdash})$, where $\mathbf{a}^{\vdash} \in \mathbb{R}^n$. An element $a_i^{\vdash}$ represents the deviation from the expected value of a TKPI $k_i^{\vdash}$ of a TKPI vector $\mathbf{k}^{\vdash} = (k_1^{\vdash}, \ldots, k_n^{\vdash})$. In addition, the TKPI anomaly vectors for $\dot{\mathbf{k}}^{\vdash} = (\dot{k}_1^{\vdash}, \ldots, \dot{k}_n^{\vdash})$ and $\mathring{\mathbf{k}}^{\vdash} = (\mathring{k}_1^{\vdash}, \ldots, \mathring{k}_n^{\vdash})$ are denoted as $\dot{\mathbf{a}}^{\vdash} = (\dot{a}_1^{\vdash}, \ldots, \dot{a}_n^{\vdash})$ and $\mathring{\mathbf{a}}^{\vdash} = (\mathring{a}_1^{\vdash}, \ldots, \mathring{a}_n^{\vdash})$, respectively.

Furthermore, the TKPI anomaly level vectors give us the TKPI anomaly level vector space. It is defined as follows:

Definition 7.2 (TKPI anomaly level vector space)**.** The collection of TKPI anomaly vectors is called the TKPI anomaly level vector space, and is denoted as $A^{\vdash} = \{\mathbf{a}_1^{\vdash}, \ldots, \mathbf{a}_{|\Sigma|}^{\vdash}\}$, where $|A^{\vdash}| = |K^{\vdash}|$, $A^{\vdash} \subset \mathbb{R}^{|\Sigma|}$, and Σ is the set of all cells. In addition, the TKPI anomaly vector spaces of all static cells $\dot{\Sigma}$ and on-demand cells $\mathring{\Sigma}$ are denoted as $\dot{A}^{\vdash} = \{\dot{\mathbf{a}}_1^{\vdash}, \ldots, \dot{\mathbf{a}}_{|\dot{\Sigma}|}^{\vdash}\}$ and $\mathring{A}^{\vdash} = \{\mathring{\mathbf{a}}_1^{\vdash}, \ldots, \mathring{\mathbf{a}}_{|\mathring{\Sigma}|}^{\vdash}\}$, respectively.

In contrast to the anomaly vectors used by the CM verification process, the computation of the TKPI anomaly level vector depends on the cell type. As stated in Section 5.3.3, only static cells have a profile. As a result, the TKPI anomaly level $\dot{\mathbf{a}}^{\vdash}$ of a cell $\dot{\sigma} \in \dot{\Sigma}$ can be computed in the same way as discussed in Section 6.1. That is, there is a function $\dot{\varphi}$ (cf. Equation 7.1) which takes a TKPI profile $\dot{\mathbf{p}}^{\vdash} \in \dot{P}^{\vdash}$, the current TKPI vector $\dot{\mathbf{k}}^{\vdash} \in \dot{K}^{\vdash}$ and returns a TKPI anomaly level vector $\dot{\mathbf{a}}^{\vdash} \in \dot{A}^{\vdash}$.

$$\dot{\varphi} \colon \dot{P}^{\vdash} \times \dot{K}^{\vdash} \rightarrow \dot{A}^{\vdash} \tag{7.1}$$

In addition, it is possible to have different profile types, e.g., such that specify the usual weekday behavior and such that define the behavior of the network during the weekend. Note that a particular example of how to implement an anomaly level function has already been introduced in Section 6.1.

However, function $\dot{\varphi}$ cannot be applied in the case of on-demand cells since they do not have a profile. Section 3.2.5 highlights and discusses the reasons for that. Hence, function $\dot{\varphi}$ is modified in such a way that it no longer requires a profile to calculate the

anomaly level. As shown in Equation 7.2, it takes three arguments. First, it takes ν TKPI profiles $\{\dot{\mathbf{p}}_1^{\vdash}, \ldots, \dot{\mathbf{p}}_\nu^{\vdash}\}$. Each of those profiles originates from a static neighbor of the given on-demand cell. Here, ν marks the number of selected neighbors. Second, the function takes the same number of TKPI vectors $\{\dot{\mathbf{k}}_1^{\vdash}, \ldots, \dot{\mathbf{k}}_\nu^{\vdash}\}$, which are compared against the selected TKPI profiles. Third, the function takes the most recent TKPI vector $\mathring{\mathbf{k}}^{\vdash} \in \mathring{K}^{\vdash}$ of the on-demand cell.

$$\mathring{\phi}\colon \{\dot{\mathbf{p}}_1^{\vdash}, \ldots, \dot{\mathbf{p}}_\nu^{\vdash}\} \times \{\dot{\mathbf{k}}_1^{\vdash}, \ldots, \dot{\mathbf{k}}_\nu^{\vdash}\} \times \mathring{K}^{\vdash} \rightarrow \mathring{A}^{\vdash} \tag{7.2}$$

Let us give an example of function $\mathring{\phi}$. For simplicity reasons, let us assume that the cell load is the only TKPI which is taken into account. Hence, the resulting TKPI anomaly vector $\mathring{\mathbf{a}}^{\vdash} \in \mathring{A}^{\vdash}$ contains only one element, namely the deviation from the expected load. Let us call this element the load anomaly level. The load anomaly level of an enabled on-demand cell can be computed as the weighted sum of the load anomaly levels of its static neighbors. The weight itself is the UE ratio, i.e., the number of UEs served by a neighboring cell divided by the total number of UEs within the area. In order to compute the load anomaly anomaly level of a disabled on-demand cell, techniques from regression analysis can be utilized.

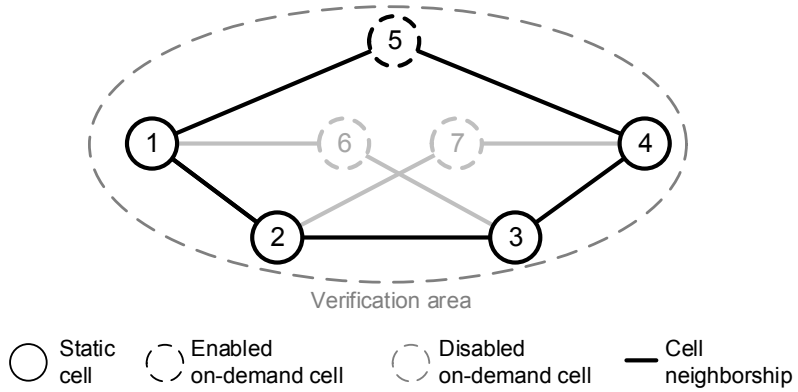
7.2 Topology Verification Graph

Similarly to the CM verification process, the process of verifying topology changes operates on verification areas, as given in Definition 5.3. The exact strategy for selecting those has been introduced in Section 5.2.2. However, in order to use the Steiner tree algorithm, each verification area, that has been generated before starting the process assessment phase (cf. Figure 7.1), is represented by a *topology verification graph*. The graph itself is defined as follows:

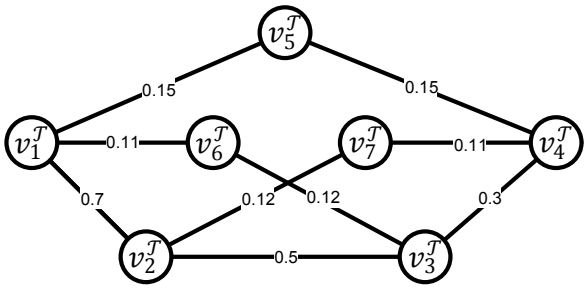
Definition 7.3 (Topology verification graph)**.** A topology verification graph is an undirected edge weighted $G^{\mathcal{T}} = (V^{\mathcal{T}}, E^{\mathcal{T}}, d^{\mathcal{T}})$, where $V^{\mathcal{T}}$ is a set of vertexes, $E^{\mathcal{T}}$ a set of edges $V^{\mathcal{T}} \times V^{\mathcal{T}}$, and $d^{\mathcal{T}}$ an edge weight function. A vertex $v^{\mathcal{T}} \in V^{\mathcal{T}}$ represents an on-demand or a static cell, whereas an edge $(v_i^{\mathcal{T}}, v_j^{\mathcal{T}}) \in E^{\mathcal{T}}$ a neighbor relation between two cells. Function $d^{\mathcal{T}}$ is defined as $E^{\mathcal{T}} \rightarrow \mathbb{R}_{\geq 0}$.

In particular, $d^{\mathcal{T}}$ computes the edge weight by taking into account the position of the cells represented by $v_i^{\mathcal{T}}$ and $v_j^{\mathcal{T}}$ in $\mathbb{R}^n$. Thereby, it estimates the distance between TKPI anomaly vectors of cells represented by $v_i^{\mathcal{T}}$ and $v_j^{\mathcal{T}}$. Examples are the Euclidean or Manhattan distance.

Figures 7.2(a) and 7.2(b) depict an example of forming $G^{\mathcal{T}}$. The first figure shows a network that consists of four static cells (cells 1-4) and three on-demand cells (cells 5-7). The neighbor relations are visualized by the lines connecting the cells. Initially, cells 6 and 7 are disabled whereas cell 5 is switched on. Note that all seven cells are part of the same verification area. The second figure visualizes the respective topology verification



(a) Static cells (1-4), enabled on-demand cell (5) and disabled on-demand cells (6,7)



(b) Topology verification graph $G^{\mathcal{T}} = (V^{\mathcal{T}}, E^{\mathcal{T}}, d^{\mathcal{T}})$

Figure 7.2: An example of forming the topology verification graph. The network is comprised of four static cells, three on-demand cells, and nine neighbor relations.

graph $G^{\mathcal{T}}$. Each of the seven cells is represented by a vertex $v_i^{\mathcal{T}}$, where index i equals the cell identifier. In addition, the edge weights are computed by a function $A^{\vdash} \times A^{\vdash} \rightarrow [0; 1]$. In the case of two enabled cells, the edge represents the TKPI anomaly level that is experienced by the two cells. In particular, 0 and 1 are an indication for a low and high TKPI anomaly level, respectively. Should, however, one of the cells be disabled, the edge shows how much the TKPI anomaly level between the cells can be improved if the turned off cell is switched on.

7.3 The Steiner Tree-Based Verification Algorithm

The Steiner tree problem itself [GHNP01] is an NP-complete problem which is why in practice heuristics are most commonly used. A well known algorithm is the one introduced by Kou, Markowsky and Berman [KMB81]. The algorithm itself forms an MST out of an input graph by potentially including extra vertexes, referred to as Steiner points. It calls twice an MST algorithm and once an algorithm that computes the shortest path between two points. Algorithm 2 lists the pseudo-code of the topology verification approach that is based on the Steiner tree algorithm.

At the beginning, it takes as input the topology verification graph $G^{\mathcal{T}} = (V^{\mathcal{T}}, E^{\mathcal{T}}, d^{\mathcal{T}})$, as described in Section 7.1. Then, the Steiner points are selected (line 1). Their set is denoted as $V^{\mathcal{P}}$, where $V^{\mathcal{P}} \subset V^{\mathcal{T}}$. The remaining vertexes $V^{\mathcal{F}}$, i.e., the complement $V^{\mathcal{T}} \setminus V^{\mathcal{P}}$, are set as fixed terminals (line 2). Initially, all on-demand cells are marked as Steiner points whereas all static cells are defined as terminals.

It should be also noted that this selection strategy applies only if the algorithm is triggered for the very first time, that is, we do not have any information regarding previous states. However, an alternative strategy has to be considered if the algorithm has already been triggered, and some on-demand cells have been turned on. In particular, some on-demand cells may become terminals if their operation is required in the future. Section 7.4 goes into more detail.

Before continuing, let us give an example based on the topology depicted in Figure 7.2. The Steiner point and terminal selection is visualized in Figure 7.3(a). Note that static cells have an ID between 1 and 4 whereas on-demand cells an ID within the range of 5 and 7. Hence, the sets of terminals and Steiner points are selected as follows: $V^{\mathcal{F}} = \{v_1^{\mathcal{F}}, v_2^{\mathcal{F}}, v_3^{\mathcal{F}}, v_4^{\mathcal{F}}\}$ and $V^{\mathcal{P}} = \{v_5^{\mathcal{P}}, v_6^{\mathcal{P}}, v_7^{\mathcal{P}}\}$.

As a next step, an undirected distance graph $D_G(V^{\mathcal{F}})$ is formed (line 3). This new graph consists only of the vertexes that have been marked as terminals. Moreover, the edge wights within this graph equal the costs given by the shortest paths between the terminals in $G^{\mathcal{T}}$. The algorithm that is particularly used here is Dijkstra's algorithm [Dij59, CLRS09]. Figure 7.3(b) visualizes the outcome after applying this step. The graph consists of the four terminal nodes, each being connected with the other.

Next, the newly formed graph $D_G(V^{\mathcal{F}})$ is transformed to an MST which is denoted as T_{D_G} (line 4). The MST algorithm being used here is Kruskal's algorithm [Kru56, CLRS09]. In the case of multiple MSTs, an arbitrary one is selected, as presented between lines 5 and 7. Figure 7.3(c) shows the MST that has been formed out of $D_G(V^{\mathcal{F}})$. In total, three edges have been removed: $(v_1^{\mathcal{F}}, v_4^{\mathcal{F}})$, $(v_1^{\mathcal{F}}, v_2^{\mathcal{F}})$, and $(v_2^{\mathcal{F}}, v_3^{\mathcal{F}})$.

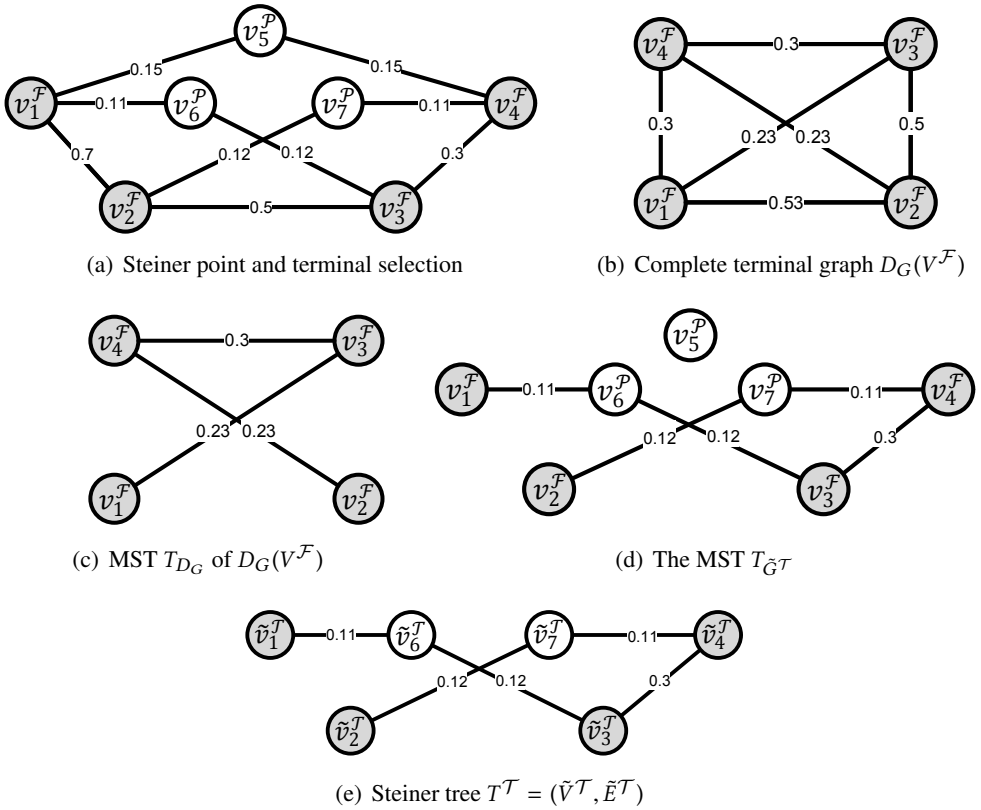
Afterwards, the formed MST is transformed to a graph $\tilde{G}^{\mathcal{T}}$ by replacing each edge by the corresponding shortest path from $G^{\mathcal{T}}$ (line 8). Furthermore, $\tilde{G}^{\mathcal{T}}$ is transformed to an MST $T_{\tilde{G}^{\mathcal{T}}}$ (line 9). Similarly to the previous step, the MST is formed by triggering Kruskal's algorithm. Those two steps are summarized in Figure 7.3(d). It should be noted that $v_5^{\mathcal{P}}$ is not required for the formation of the tree.

Finally, the Steiner tree $T^{\mathcal{T}} = (\tilde{V}^{\mathcal{T}}, \tilde{E}^{\mathcal{T}})$ is formed by continuously removing non-terminal leaves from $T_{\tilde{G}^{\mathcal{T}}}$ or non-terminals that remained disconnected (line 10). In addition, the set of unnecessary Steiner points is formed by taking the complement of the set of Steiner points $V^{\mathcal{P}}$ and the set $\tilde{V}^{\mathcal{T}}$ (line 11). In line 12, the set of required Steiner points is formed. Figure 7.3(e) depicts the end result. The inclusion of Steiner points $v_6^{\mathcal{P}}$ and $v_7^{\mathcal{P}}$ leads to a spanning tree that is shortest in length.

Algorithm 2: Steiner tree-based verification algorithm

Input: Undirected, edge weighted topology verification graph $G^{\mathcal{T}} = (V^{\mathcal{T}}, E^{\mathcal{T}}, d^{\mathcal{T}})$
Result: Steiner tree $T^{\mathcal{T}}$ out of $G^{\mathcal{T}}$, a set $V^{\mathcal{U}} \subset V^{\mathcal{T}}$ of unnecessary Steiner points, and a set $V^{\mathcal{R}} \subset V^{\mathcal{T}}$ of required Steiner points

1 Select Steiner points $V^{\mathcal{P}} \subset V^{\mathcal{T}}$;
2 $V^{\mathcal{F}} \longleftarrow V^{\mathcal{T}} \setminus V^{\mathcal{P}}$;
3 Construct a complete undirected distance graph $D_G(V^{\mathcal{F}})$;
4 Compute a minimum spanning tree T_{D_G} of $D_G(V^{\mathcal{F}})$;
5 **if** *Multiple* T_{D_G} *present* **then**
6 Select an arbitrary minimum spanning tree;
7 **end**
8 Form $\tilde{G}^{\mathcal{T}}$ by replacing each edge in T_{D_G} by the corresponding shortest path from $G^{\mathcal{T}}$;
9 Form a minimum spanning tree $T_{\tilde{G}^{\mathcal{T}}}$ from $\tilde{G}^{\mathcal{T}}$;
10 Compute $T^{\mathcal{T}} = (\tilde{V}^{\mathcal{T}}, \tilde{E}^{\mathcal{T}})$ by continuously removing leaves from $T_{\tilde{G}^{\mathcal{T}}}$ that are $\notin V^{\mathcal{F}}$;
11 $V^{\mathcal{U}} \longleftarrow V^{\mathcal{P}} \setminus \tilde{V}^{\mathcal{T}}$;
12 $V^{\mathcal{R}} \longleftarrow V^{\mathcal{P}} \setminus V^{\mathcal{U}}$;



(a) Steiner point and terminal selection

(b) Complete terminal graph $D_G(V^{\mathcal{F}})$

(c) MST T_{D_G} of $D_G(V^{\mathcal{F}})$

(d) The MST $T_{\tilde{G}^{\mathcal{T}}}$

(e) Steiner tree $T^{\mathcal{T}} = (\tilde{V}^{\mathcal{T}}, \tilde{E}^{\mathcal{T}})$

Figure 7.3: Example of applying the Steiner tree-based verification algorithm. The four static cells (terminals) are colored in gray whereas the three on-demand cells (Steiner points) in white.

7.4 Topology Correction and Steiner Point Assessment

After triggering the Steiner tree algorithm, we get a Steiner tree $T^{\mathcal{T}}$ out of the topology verification graph $G^{\mathcal{T}}$. This tree has been formed by filtering out unnecessary Steiner points and using only those that minimize its total weight. Hence, cells represented by Steiner points that are left over are seen as a set of *unnecessary on-demand cells* $\mathring{\Sigma}^{\mathcal{U}} \subseteq \mathring{\Sigma}$, whereas the remaining ones as a set of *required on-demand cells* $\mathring{\Sigma}^{\mathcal{R}} \subseteq \mathring{\Sigma}$. Formally, those sets are specified as follows:

Definition 7.4 (Unnecessary and required on-demand cells)**.** The vertexes $V^{\mathcal{U}}$ that remain unused to form the Steiner tree $T^{\mathcal{T}} = (\tilde{V}^{\mathcal{T}}, \tilde{E}^{\mathcal{T}})$ for a given topology verification graph $G^{\mathcal{T}} = (V^{\mathcal{T}}, E^{\mathcal{T}}, d^{\mathcal{T}})$, i.e., $V^{\mathcal{U}} \subset V^{\mathcal{T}}$ and $V^{\mathcal{U}} \cap \tilde{V}^{\mathcal{T}} = \emptyset$, represent the set of unnecessary on-demand cells $\mathring{\Sigma}^{\mathcal{U}}$. On the contrary, the set of Steiner points $V^{\mathcal{R}}$ used to form the tree, i.e., $V^{\mathcal{R}} \subset V^{\mathcal{T}}$ and $V^{\mathcal{R}} \subset \tilde{V}^{\mathcal{T}}$ constitute the set of required on-demand cells $\mathring{\Sigma}^{\mathcal{R}}$.

Figure 7.4 represents the final outcome after enabling the required and disabling the unnecessary on-demand cells. Note that the initial network topology is presented in Figures 7.2 and 7.3. Within the given verification area, cell 5 is disabled since the vertex representing it is not required to form the Steiner tree. Cells 6 and 7 are needed and are, therefore, turned on.

However, if we strictly follow the up to now outlined strategy, we may face a similar problem as discussed in Section 6.2, namely generating unnecessary corrective actions in the case of fluctuating PM data. For instance, if a rather large group of UEs leaves and enters the same area, it would be undesirable to disable and shortly after that enable nearby on-demand cells. For this reason, enabled on-demand cells are evaluated by the Steiner point assessment function ι, as given in Equation 7.3.

$$\iota \colon \mathring{\Sigma}^{\mathcal{R}} \times \mathring{K}^{+} \rightarrow V^{\mathcal{P}} \cup \emptyset \tag{7.3}$$

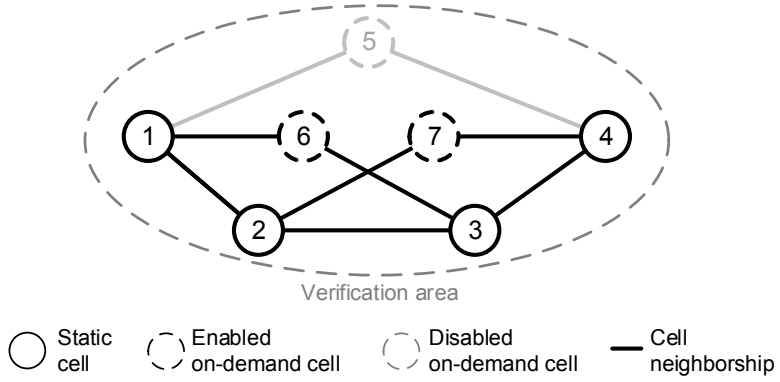


Figure 7.4: Corrective topology actions generated by the Steiner tree-based verification algorithm. Turning on on-demand cells 6 and 7, and turning off on-demand cell 5.

This function permits the exclusion of an enabled on-demand cell $\mathring{\sigma}^{\mathcal{R}} \in \mathring{\Sigma}^{\mathcal{R}}$ from the Steiner point set $V^{\mathcal{P}}$. The decision itself depends on the TKPI vector $\mathring{\mathbf{k}}^{\vdash} \in \mathring{K}^{\vdash}$ of the on-demand cell. For instance, if it is continuously experiencing a low load, we may set it as a Steiner point. On the contrary, continuously reporting a high load may mean that the cell will be required in the future, i.e., we may consider the cell as a terminal during the next iteration (cf. line 1 of Algorithm 2).

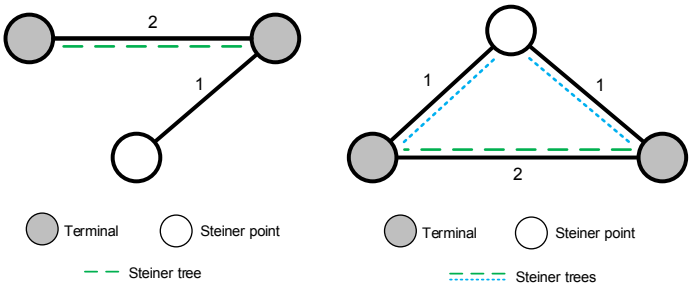
In the same connection, the question arises how to penalize on-demand cells that are continuously toggled due to fluctuating PM data. For example, a large UE group enters the network for a short time frame which temporary triggers a high load on the static cells. Consequently, the chance of turning on an on-demand cell increases. This issue is solved by artificially extending the edge weight between vertexes that represent an on-demand and a static cell in the topology verification graph $G^{\mathcal{T}}$, i.e., by multiplying it by a certain factor. As a result, the likelihood of selecting those Steiner points in $T^{\mathcal{T}}$ decreases.

7.5 Analysis of the Steiner Tree-Based Verification Approach

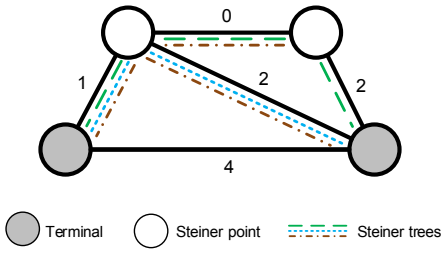
Modeling the topology verification problem, as described between Sections 7.1 and 7.4 creates a set of challenges that are not addressed by the Steiner tree algorithm. First of all, if an on-demand cell has only one cell as neighbor, it will be added to the topology verification graph $G^{\mathcal{T}}$ as a leaf. As a result, it will be considered as a leaf by the Steiner algorithm which means that the Steiner point representing the cell will be excluded in any case from the end result. It is caused by the fact that adding a Steiner point as leaf results in a spanning tree that is not minimal in length. Figure 7.5(a) visualizes this problem. No matter what the cost between the terminal and the Steiner point is, the latter one gets always excluded. Hence, such on-demand cells have to be either marked as terminals from the very beginning or be evaluated separately.

Second, the Steiner tree problem does not provide an explicit strategy for selecting Steiner trees of equal length. Figure 7.5(b) depicts such a scenario. The weight of the spanning tree that includes the Steiner point equals the weight of the spanning tree that consists only of the two terminal nodes. As a result, there are two possible outcomes, i.e., two permissible sets of topology corrective actions. Activating the on-demand cell would be the more aggressive strategy.

Third, we may have an edge weight of zero between Steiner points. Usually, this happens when we have two or more disabled on-demand cells in the network. Since neither of the cells are serving UEs, nor are experiencing any workload, there is a chance for having a cost of zero between the vertexes representing them in $G^{\mathcal{T}}$. Figure 7.5(c) visualizes this problem in the case of two Steiner points. As it can be seen, there are three permissible Steiner trees: two that include both Steiner points and one that is formed by taking only one of the points. Hence, we have two valid outcomes for generating topology corrective actions: turning only one or both on-demand cells. Similarly to the previous problem, activating more than the minimum number of required on-demand



(a) Steiner points as leaves (b) Equally weighted Steiner trees



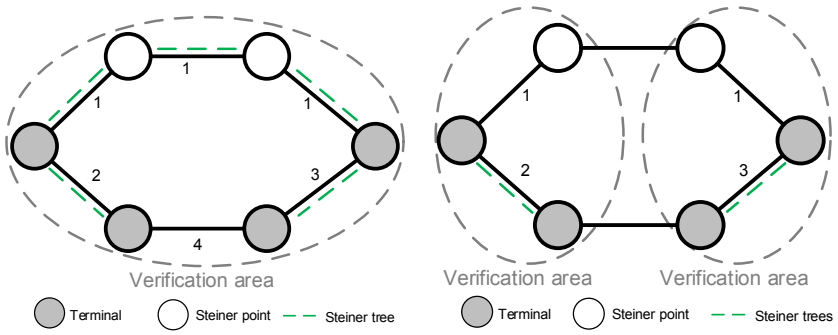
(c) Zero cost edges

Figure 7.5: Challenges when applying the Steiner tree algorithm: exclusion of Steiner points defined as leaves, and multiple solutions due to equally weighted Steiner trees or zero cost edges.

cells is seen as an aggressive approach.

Fourth, the verification area selection plays a crucial role for the end result. Let us consider the example shown in Figure 7.6(a). The graph consist of four terminals and two Steiner points. Furthermore, all nodes and edges are part of the same verification area which, as a result, means that the Steiner tree algorithm will be called only once, namely for that particular area. As shown, the algorithm adds both Steiner points to the end result which has the consequence of switching on all on-demand cells. On the contrary, if we form two smaller verification areas, as shown in Figure 7.6(b), we will get two simplified topology verification problems. Each of those problems is solved by the Steiner tree algorithm separately, i.e., we have two input graphs for each of which the Steiner tree algorithm is called. As the example shows, the Steiner points are considered as leaves in the two input graphs which may lead to the first problem outlined in the beginning of this section. Here, the on-demand cells will be disabled since the points were not required to form the Steiner tree.

However, the selection of a larger verification area by including more network entities, may induce another set of challenges. First, the verification problem that needs to be solved becomes more complex due to the increased vertex number as well as the number of edges that need to be taken into account. Second, the number of edges increases, for which we would need to find a unified metric. The edge costs depend on the TKPIs within the whole verification area and represent how a pair of two cells performs with



(a) Single verification area (b) Multiple verification areas

Figure 7.6: Impact of the verification area selection on the Steiner tree algorithm. Changing the size of the verification area may lead to a different set of corrective actions.

respect to the remaining cells within the area. As a result, the exclusion or inclusion of cells will lead to a change of the edge costs.

7.6 Summary

This chapter presented the process that verifies topology changes, as initially introduced in Section 5.1. It has the purpose of generating topology corrective actions that either enable or disable cells. The approach itself is based on the Steiner tree algorithm, as the following research contribution outlines:

O1.2: Model and design a verification process that assesses CM changes as well as changes made in the network topology.
Q: How to model the topology verification problem?
A: The topology verification problem is modeled as a Steiner tree problem. The latter one is a combinatorial optimization problem that tries to reduce the length of an MST by adding extra vertexes and edges to the initial edge weighted graph. Those additional vertexes are referred to as Steiner points whereas the initial nodes are called terminals. In general, Steiner points represent cells that can be turned on or off during their operation (called on-demand cells) whereas terminals mark cells that remain always switched on (referred to as static cells). Based on whether a cell is used as a Steiner point to form the tree, it is decided if and how to consider it while generating the corrective action plan.

In this chapter, also the impact of the verification area selection on the topology verification process has been discussed. Concretely, the following research contribution has been made:

O2.1: Fragmentation of the network and specifying the scope of verification.
Q: Does the verification area selection affect the Steiner tree-based verification algorithm?
A: Yes. It is of high importance how the verification area is selected. The larger a verification area is, the more complex it becomes to solve the Steiner tree problem. The input graph optimized by the Steiner algorithm increases in size for which we would need to find a unified cost metric. On the other side, the smaller a verification area is, the more likely it becomes Steiner points to become leaves in the input graph. Hence, they will never be selected to form the Steiner tree.

Furthermore, the issues that emerge when having incomplete profiles have been addressed. Also, the cell state has been specified in the case of topology verification. In particular, the following research contributions have been made:

O2.2: Definition of the verification state.
Q: How to model the state of a cell while verifying topology changes?
A: The state of a cell is represented by its TKPI anomaly vector (cf. Definition 7.1). Similarly to the CM verification process (cf. Chapter 6), each element of the vector represents the deviation from the experted TKPI value. The anomaly vector computation is modeled as a function (cf. Equation 7.1) that takes a cell's profile and the current TKPI vector. Also, the collection of all TKPI anomaly level vectors is referred to as the TKPI anomaly level vector space (cf. Definition 7.2).

Q: How to compute a TKPI anomaly level when profiles are incomplete?
A: The computation is done by function $\hat{\phi}$ (cf. Equation 7.2). Since an on-demand cell does not have a valid profile, the anomaly vector computation is modeled as a function that takes the TKPI profiles and TKPIs of its static neighbors. Furthermore, the TKPIs an on-demand cell is generating are considered as well.

Q: How to define the relation between cells when verifying topology changes?
A: The relation between cells is represented by an undirected edge weighted topology verification graph (cf. Definition 7.3). Vertexes represent cells whereas edges neighbor relations. The weighting function is taking into account the position of the cells represented in $\mathbb{R}^n$, as given by their TKPI anomaly level vectors.

Q: Do cells change their state when using the Steiner tree-based verification algorithm?
A: Yes. The Steiner tree-based verification algorithm allows on-demand cells to be selected not only as Steiner points, but also as terminals. Marking an on-demand cell as a Steiner point means it that *may* get included in the Steiner tree, whereas

setting it as a terminal has the consequence that it *must* be used while forming the tree. Hence, on-demand cells that are going to be required in the future are defined as terminals. The actual decision is made by the Steiner point assessment function ι (cf. Equation 7.3).

In this chapter, a contribution to the objective that studies the verification process duration has also been made.

O2.3: Estimate the duration of the verification process.
Q: Is the Steiner tree-based verification algorithm able to penalize on-demand cells?
A: The Steiner tree-based verification algorithm permits the penalization of on-demand cells that get switched on or off due to fluctuating PM data. It is achieved by increasing the weights of the edges leading to a Steiner point. Hence, the probability of selecting it when forming the Steiner tree decreases. Nonetheless, extending the edges may also delay the process of providing a corrective topology action as they will not be considered by the Steiner tree algorithm.

Finally, a contribution to the remaining topology verification research objectives has been made. The following answers further highlight the achievements:

O4.1: Specify and define the uncertainties caused by topology changes.
Q: What is the relation between the verification of topology changes and the Steiner tree problem?
A: The verification of topology changes is modeled as a Steiner tree optimization problem. The outcome of the Steiner tree algorithm yields the set of topology corrective actions. In particular, cells required to form the Steiner tree get or remain enabled, whereas the unnecessary ones get disabled.

O4.2: Study and evaluate the impact of topology changes on the process of verifying configuration changes, as well as identify the necessary conceptual changes of a verification process.
Q: How to model and select cell profiles when the network topology is changing?
A: The Steiner tree-based verification approach distinguishes between on-demand and static cells. Only static cells have a profile which specifies the expected range of the monitored TKPIs and is required to calculate the TKPI anomaly levels (cf. Equation 7.1). On-demand cells on the other side do not have a profile and compute the anomaly levels by considering the TKPIs and profiles of their neighbors as well as their current TKPIs (cf. Equation 7.2).

O4.3: Enable topology verification, resolve uncertainties, and provide corrective actions.
Q: How does the topology verification process enable the verification of topology changes and does it prevent configuration changes of being blamed by mistake?
A: As mentioned above, the Steiner tree-based verification approach uses the outcome of the Steiner tree algorithm to enable or disable cells, also referred to as corrective topology actions. Furthermore, it is triggered before the CM verification process, i.e., cells which are turned on or off by the topology verification process are not assessed by the process that verifies configuration changes. Thereby, the likelihood of uncertainties to emerge while performing verification decreases as the topology verification and CM verification problem are solved by two processes, each knowing the outcome of the other.

Q: For which verification-related problems additional care must be taken?
A: By default, the Steiner algorithm will always exclude Steiner points that are added to the initial input graph as leaves. Hence, on-demand cells having just one static cell as neighbor are always excluded from the Steiner tree, i.e., they are turned off. As a result, such cells must be evaluated separately. Furthermore, the algorithm does not provide a strategy for selecting MSTs of equal length, i.e., there might be two or more valid solutions. It also does not provide a strategy for handling zero-weight edges, in particular, between Steiner points. Such edges result in multiple permissible topology corrections.

Part IV

Concept Implementation and Evaluation

Chapter 8

Evaluation Environment

In this chapter, an introduction to the simulation as well as real data environment is given. Both systems are utilized during the evaluation of the presented verification concept. Note that the concept was comprehensively described in Chapters 5, 6, and 7. The evaluation on the other side can be found in Chapter 10.

Here, the properties of the simulator, the simulated network topology, the UE mobility as well as the available SON functions are described. Section 8.1 is devoted simulation related topics. Section 8.2, however, is dedicated to the real data set exported by an LTE network. Concretely, the PM and CM parameters, the data export frequency, and the active SON functions are discussed in detail.

Published work An overview of the environments described in this chapter can be also found in [TNSC14a] (real data study) and [TATSC16c] (simulation study). Compared to those papers, this chapter gives a more comprehensive description of the simulation and real data environment.

8.1 Simulation Environment

The simulation environment, that is used for the evaluation of the presented verification concept, is called the SON Simulation System (S3). It consist of five components: an LTE radio network simulator, a simulator parser, a SON function collection, a SON function coordinator, and the already introduced verification process (cf. Chapters 5 to 7). Figure 8.1 visualizes all components. As shown, the components can only interact when a simulation scenario is defined. It specifies the configuration parameters, the network topology, and all other rules and policies that must be set up. In the upcoming sections each of those components will be introduced. It should be noted, though, that the implementation of the verification process is presented in detail in Chapter 9 and will be, therefore, not further discussed here.

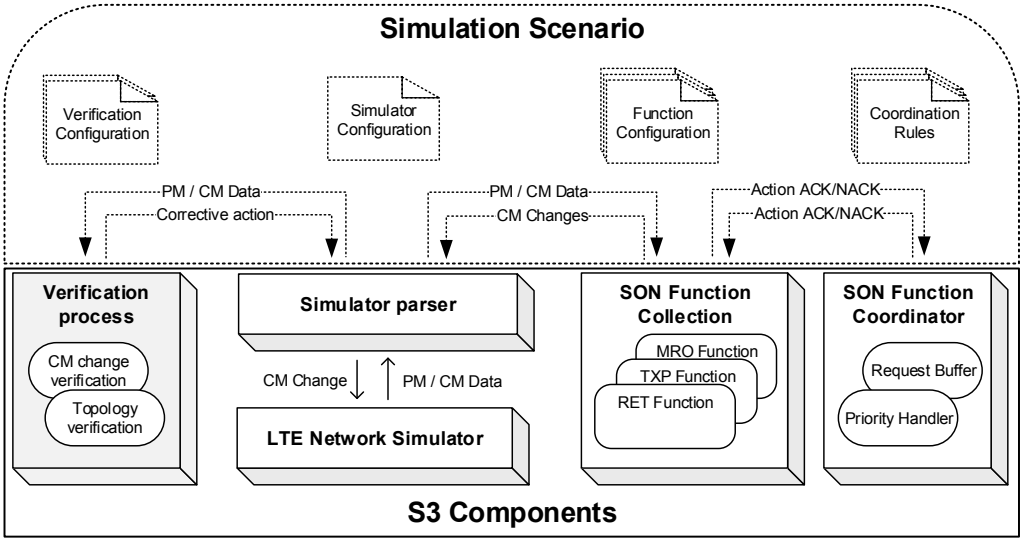


Figure 8.1: Overview of the simulation environment. It consist of five components which may interact with each other. The interaction itself is defined by the simulation scenario.

8.1.1 LTE Network Simulator and Parser

The LTE network simulator is part of the SON simulator/emulator suite [NSN09]. It simulates only the EUTRAN (cf. Section 2.1.3), i.e., the EPC (cf. Section 2.1.4) is omitted from the simulation test runs. In the upcoming sections, the properties of the simulator are described.

8.1.1.1 Simulation Time

The LTE simulator performs continuous simulation by tracking the changes in the network over time. The time itself is divided into time slices which are referred to as *simulation rounds*. At the end of a simulation round all the collected PM data is exported. The data set itself consists of 14 cell KPIs which are later described in Section 8.1.1.2. Those KPIs can be monitored by any entity, e.g., the active SON functions or the verification process. Hence, those entities are usually becoming active at the end of a simulation round, i.e., CM changes are made only after the completion of a round. Note that the complete list of CM parameters is given in Section 8.1.1.3. Also, due to the batch-like export of PM data, a simulation round can be seen as an equivalent to a PM granularity period (cf. Section 2.3). Figure 8.2 visualizes an exemplary representation of the simulation time. As presented, SON functions get active after the completion of a simulation round and, based on their objectives, trigger CM changes.

During a simulation round, users actively use the mobile network. Section 8.1.1.4 lists the properties of all UE groups, e.g., placement, mobility model, and Constant Bit Rate (CBR) requirement. Furthermore, topology specific details, e.g., neighbor relations and eNB locations, are provided in Section 8.1.1.5.

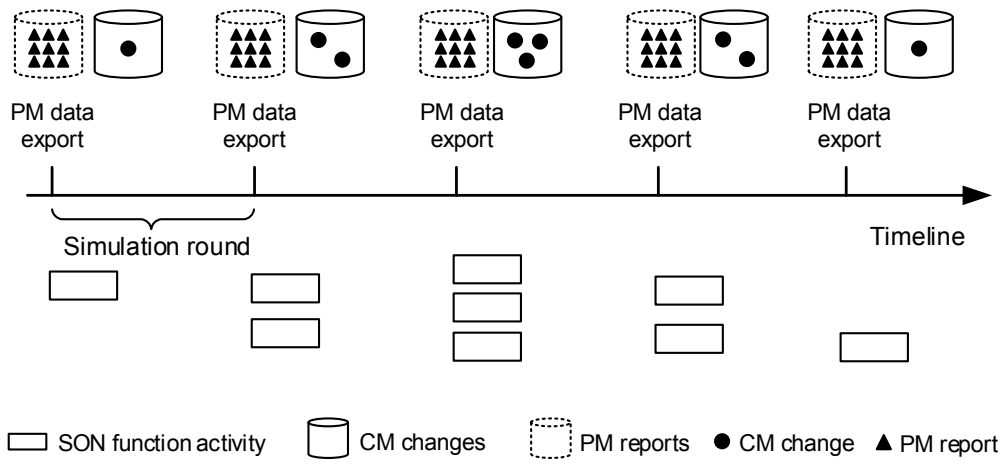


Figure 8.2: Representation of the simulation time. PM data is exported only at the end of a simulation round. Based on the PM reports, SON algorithms can get active and change CM parameters.

8.1.1.2 PM Data

The LTE network simulator exports the following set of cell KPIs:

- `KPI_THR`: Average cell throughput in bits per second.
- `KPI_RLF`: This KPI reports the number of radio link failures per serving cell.
- `KPI_CQI`: This KPI represents the Channel Quality Indicator (CQI). S3 computes it as the weighted harmonic mean of the CQI channel efficiency. The efficiency values are listed in [3GP14a].
- `KPI_PRB_UTIL`: This KPI gives a cumulative distribution function of the cell load. It provides 10 bins that correspond to the 5, 15, 25, 35, 45, 55, 65, 75, 85, 95 percentiles of the Physical Resource Block (PRB) utilization of the given cell.
- `KPI_RSRP_MIN/MAX/MEAN`: This KPI gives the minimum, maximum and mean values of RSRP at handover time.
- `KPI_HO_ATT`: The number of handover attempts that have been made at the given cell.
- `KPI_HO_DROP`: The number of handover drops that have occurred at the given cell.
- `KPI_HO_EARLY`: This KPI counts radio link failures followed by immediate reconnection to the originating cell.
- `KPI_HO_LATE`: This KPI counts radio link failures followed by an immediate reconnection to a different neighboring cell.

- KPI_HO_PINGPONG: This KPI counts ping-pong events that occur at the given cell.
- KPI_HO_WRONGCELL: This KPI counts the handover attempts made to wrong cells.
- KPI_HOSR: The hand over success rate. It is computed by dividing the number of successful handovers by the total number of handovers.

8.1.1.3 CM Parameters

The CM parameters that can be changed during a simulation test run are the following:

- ANT_ELE_TILT: Electrical tilt value of each cell's antenna beam in degrees. The value is bounded by a minimum and a maximum electrical tilt threshold value.
- TX_POWER_CHANGE: Changes the transmission power in dBm of a cell. The value is added to the current transmission power and bounded against the minimum and maximum limits specified before starting the simulation test run.
- ANT_BEAM_STEER: Controls the beam steering angle in degrees.
- HO_OFFSET_BIAS: Gives an additional bias to the handover offset value of a pair of cell neighbors. The minimum and maximum bias can be changed at run time.

8.1.1.4 UE Groups

During the simulation test runs, up to five UE groups are allowed to actively use the network. Table 8.1 shows their size, speed, movement as well as their network resource requirements. As it can be seen, there are two UE group types. On the one hand, we have a regular user group which is active during all test runs. It consists of 1500 UEs that are uniformly distributed over the whole coverage area and actively use the services of the network. Note that the coverage map is presented in Section 8.1.1.5. On the other hand, there are four hot spot user groups which are dynamically added to or removed from network. Those groups are much smaller in size and are required for the evaluation of the Steiner tree-based verification approach. The latter one has been introduced in Chapter 7.

Table 8.1: Properties of the UE groups

UE Group	Size	Speed	Movement	CBR
Regular user group	1500 UEs	6 km/h	Random walk	175 kbps
Hot spot user group 1	150 UEs	6 km/h	Random walk	175 kbps
Hot spot user group 2	75 UEs	6 km/h	Random walk	175 kbps
Hot spot user group 3	85 UEs	6 km/h	Random walk	175 kbps
Hot spot user group 4	120 UEs	6 km/h	Random walk	175 kbps

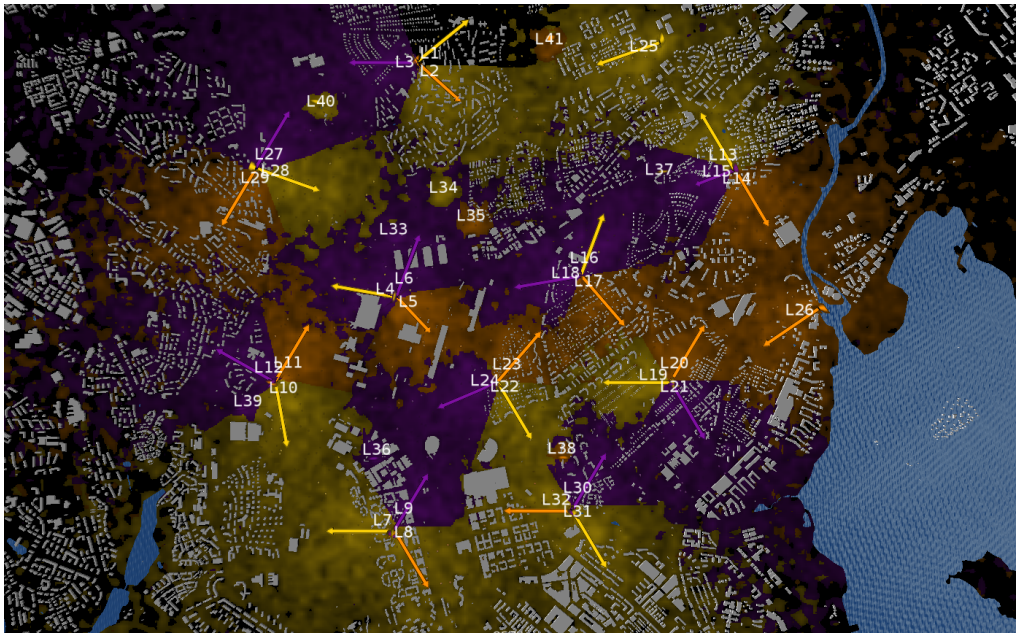


Figure 8.3: Representation of the simulated LTE network. The ID of a cell begins with the marker "L" whereas buildings are depicted by the gray shapes. The colors visualize the coverage provided by a cell.

8.1.1.5 Network Topology

The network that is simulated covers northern parts of Helsinki, Finland. Figure 8.3 shows the coverage map, including the position of the eNBs as well as the direction of each cell. The direction is visualized by differently colored arrows whereas a single coverage area is identified by the corresponding cell color. In addition, buildings are represented by gray shapes whereas UEs are depicted as colored dots within the given coverage area.

Besides the LTE macro cell layer (cell ID 1-32), the simulation system provides access to nine hot spot small cells (cell ID 33-41). As visualized in Figure 8.3, they are placed within the coverage of the following LTE macro cells: 1-4, 6, 7, 9, 10, 12, 15, 16, 18, 22, 24, 25, 27, 28, 30, and 32. Table 8.2 lists all radio parameters of the simulated LTE network.

Based on the given network setup, two neighbor relation setups are specified. On the one hand, there is a setup that consists only of the LTE macro cell layer, in particular, all 32 LTE cells. The resulting neighbor relation graph is presented in Figure 8.4. On the other hand, there is a configuration that adds to the existing LTE macro layer a second layer that consists of all available small cells. The resulting neighbor relations are visualized in Figure 8.5.

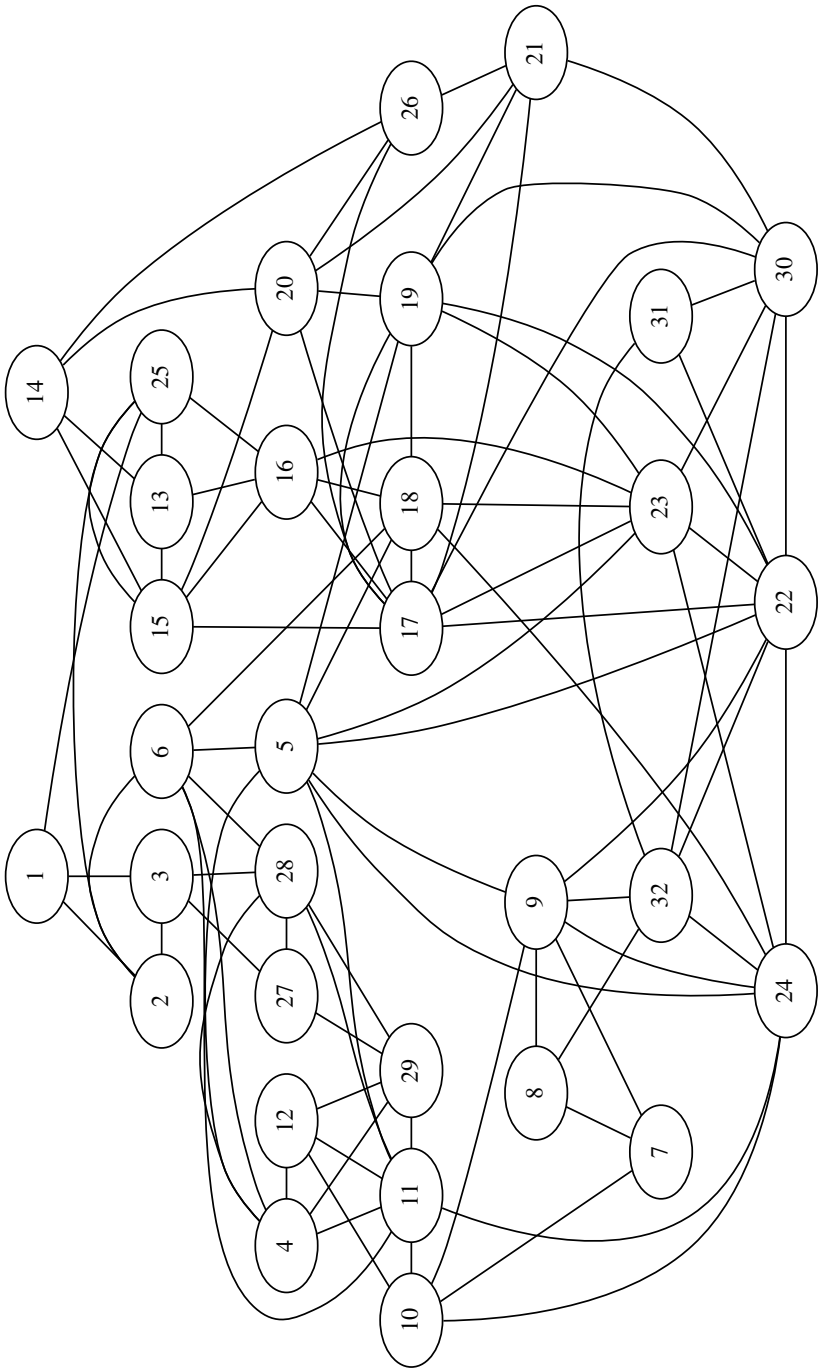


Figure 8.4: Overview of the neighbor relations in the setup consisting only of LTE macro cells. The cell IDs range between 1 and 32.

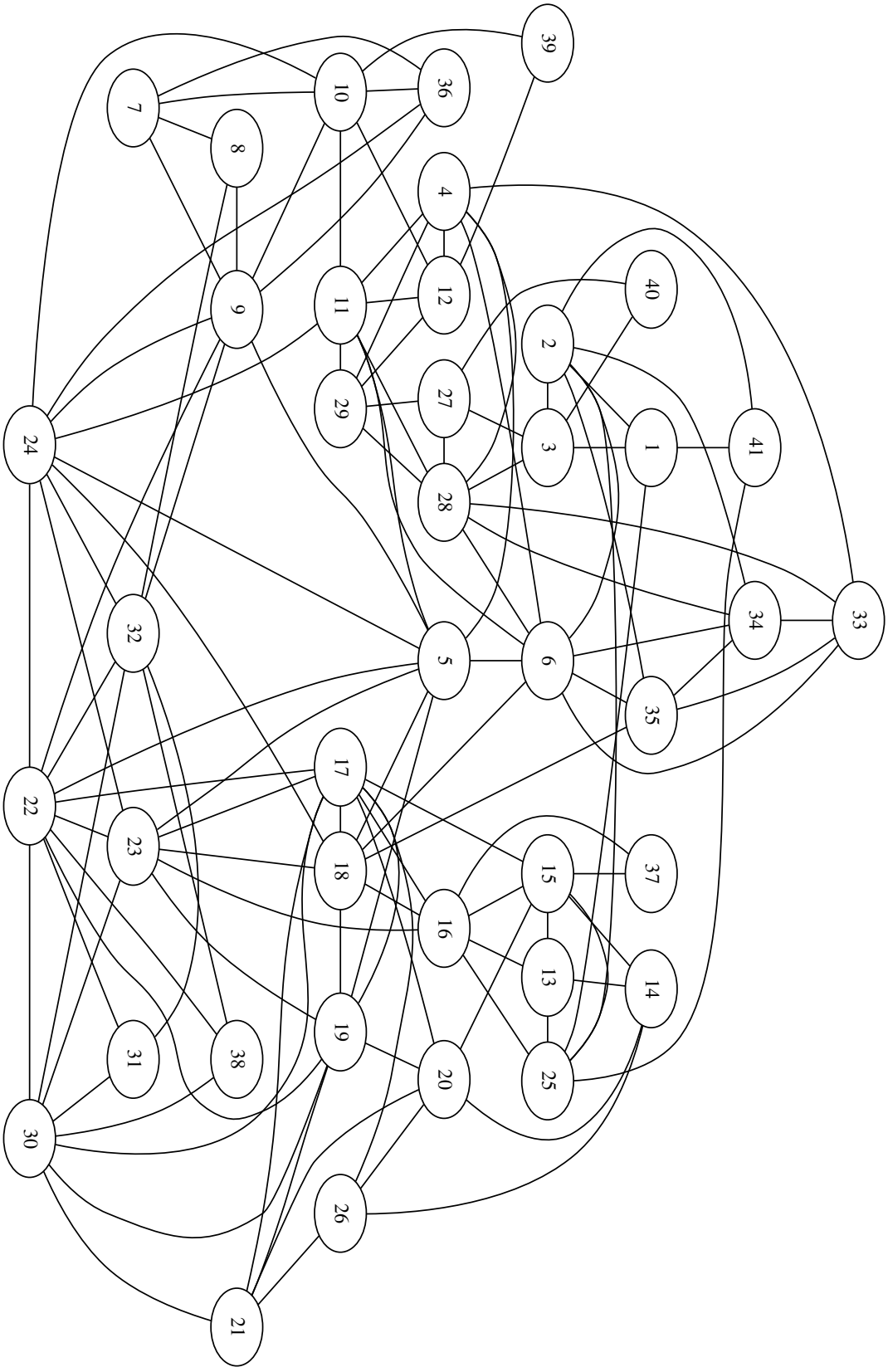


Figure 8.5: Overview of the neighbor relations in the setup consisting of LTE macro as well as small cells. Macro cell ID range: [1; 32], small cell ID range: [33; 41].

Table 8.2: Simulation environment parameter selection

	Parameter	Value
LTE radio simulator	Network frequency	2000 MHz
	Channel bandwidth	20 MHz
	Number of PRBs	100
	Shannon gap	-1.0 dBm
	Thermal noise	-114.447 dBm
	Pathloss coefficient L_a	128.1 dB
	Pathloss coefficient L_b	37.6 dB
	Path loss model	UMTS 30.03 [3GP98]
	Downlink scheduler mode	CBR mode
	Shadowing correlation distance	50.0 m
	Shadowing standard deviation	8.0 dB
	RLF model	SINR [dB] < RLF Threshold [dB]
	RLF threshold	-6.0 dB
	RLF disconnection timer	0.3 s
	RLF reconnection timer	1.0 s
	Handover hysteresis threshold	2.0 dB
	Handover ping-pong detection duration	4.0 s
	Handover states detection offset	1.0 s
	Handover timeout period	3.0 s
	Simulated time during a round	90 min
Macro cells	Antenna height of macro cells	17-20 m
	Transmission power of macro cells	46 dBm
	Antenna tilt of macro cells	0.0° - 3.5°
	Antenna gain	14 dB
	Horizontal beam angle	65°
	Vertical beam angle	9°
Small cells	Antenna height of small cells	10 m
	Horizontal beam angle	360°
	Vertical beam angle	9°
Topology	Simulated area	50 km^2
	Total macro cells	32 cells
	Total small cells	9 cells
	Neighborship degree (all cells)	5.805
	Neighborship degree (macro cells only)	5.688

8.1.2 SON Functions

As mentioned in the very beginning of this chapter, there are three SON functions that can be utilized for optimization purposes: MRO, RET, and the Transmission Power (TXP) function. All three functions fall within the self-optimization class (cf. Section 2.3.1). Note that in literature the latter two functions are also referred to as CCO-RET and CCO-TXP, respectively. Furthermore, an instance of each of those functions may run on every LTE macro cell. In the upcoming sections, the goals and the CM parameters they change are described.

8.1.2.1 MRO Function

MRO has the goal to guarantee proper mobility, i.e., appropriate cell re-selection in idle mode and proper handovers in connected mode (cf. Section 2.1.3.2). In particular, it has the following objectives [HSS11]:

- Minimization of call drops.
- Minimization of radio link failures.
- Minimization of unnecessary handovers, also called handover ping-pongs.
- Minimization of idle mode problems.

Furthermore, it tries to minimize the number of too late, too early, and handovers to wrong cells. The function that is used in this thesis achieves its tasks by altering the CIO parameter.

Note that the utilized MRO function implementation is responsible for the mobility optimization only within the same RAT, i.e., LTE, and the same frequency band, i.e., 2000 MHz.

8.1.2.2 RET Function

The RET function is a special CCO type, as defined in [HSS11, 3GP14b]. The objective of this function is to provide sufficient coverage and capacity in the network with minimal radio resources. In particular, it has the following goals:

- Improved coverage.
- Improved cell-edge bit-rate.
- Improved cell throughput.

The function used in this thesis achieves its tasks by changing the antenna tilt degree. Furthermore, it monitors KPIs that reflect the performance within the coverage area.

8.1.2.3 TXP Function

Similarly to the RET function, the TXP function is another CCO type, as described in [HSS11], i.e., it has the same objectives as the RET function. However, it tries to reach its goal by solely changing the transmission power of the antenna.

8.1.3 SON Function Coordinator

The SON coordinator performs pre-action coordination by using a batch coordination concept with dynamic priorities, as described in [RSB13]. The used coordination mechanism is designed for batch processing of SON function requests. In particular, every SON function instance has an assigned bucket and dynamic priority. The bucket initially contains a number of tokens that are decreased every time a SON function action request is accepted and increased otherwise. In case the bucket gets empty, the priority of the SON function is set to minimum. The priority of a SON function is increased again if its requests start being rejected.

In the terms of the simulation system, the coordinator collects all requests during a simulation round, determines the conflicts and sends an ACK for the requests with the highest priority and a NACK for the others.

8.2 Real Data Environment

The data set consists of PM and CM data dumps that have been exported by an LTE network between the 25^{th} of November 2013 and the 17^{th} of December 2013 [TNSC14a]. The network itself consists of 1230 eNBs. In the upcoming sections, an overview of those data sets is given. A description of the SON functions that were active during that time frame is provided as well.

8.2.1 Data Sets

8.2.1.1 PM Data

The exported PM data consists of the following eight KPIs:

- `RRC_CONN_SETUP_SR` : success rate of the elementary procedure RRC connection setup.
- `EUTRAN_ERAB_SETUP_SR_nGBR`: success rate of the elementary procedure EUTRAN Radio Access Bearer (E-RAB) setup for non GBR services.
- `ERAB_DR`: E-RAB drop rate. The rate of abnormally dropped bearers.
- `INTER_eNB_HO_SR`: inter eNB handover success rate.
- `EUTRAN_RLC_PDU_RETR_R_DL`: retransmission rate for RLC Protocol Data Units (PDUs) in downlink direction.

- EUTRAN_RLC_PDU_RETR_R_UL: retransmission rate for RLC PDUs in uplink direction.
- EUTRAN_CELL_AVA_excBLU: shows the cell availability, excluding Blocked by User (BLU) state.
- EUTRAN_INTER_eNB_X2_HO_ATT: number of inter eNB (X2-based) handover attempts.

Those KPIs were exported hourly, except on the 30^{th}, 31^{st} of November 2013 and the 6^{th}, 7^{th} of December 2013.

8.2.1.2 CM Parameters

As in every telecommunication management database, physical or logical elements of the network are represented by management objects. During the time frame, 12 daily CM snapshots were taken from the network for the following dates:

- 26^{th} - 29^{th} of November 2013: 4 snapshots
- 2^{nd} - 5^{th} of December 2013: 4 snapshots
- 9^{th} - 11^{th} of December 2013: 3 snapshots
- 13^{th} of December 2013: 1 snapshot

Those snapshots contained 141 configuration parameters. They were grouped in 6 managed object classes, as follows:

- Multi-RAT managed object: 12 parameters
- eNB managed object: 26 parameters
- Cell managed object: 86 parameters
- eNB adjacency managed object: 5 parameters
- eNB - cell adjacency managed object: 10 parameters
- Cell adjacency managed object: 2 parameters

In addition, the following events were visible in the data set:

- Managed object creation: Occurs when a managed object appears for the first time. The event also occurs when an object is recreated after a remove event.
- Managed object removal: Appears when a previously created object is missing from the snapshot of the data set.

- Managed object modification: when there is a visible change of one or more parameters.

- Unknown managed object modification: when there is no visible change in any of the dumped parameters, however, the last modification time has changed.

8.2.2 SON Functions

It is known that for the given time frame, two SON functions were deployed and were allowed to make changes: the PCI allocation and ANR function. Both functions are representatives of the self-configuration class (cf. Section 2.3.1).

8.2.2.1 PCI Function

A crucial parameter in today's LTE networks is the Physical Cell Identity (PCI). It is a low level identifier broadcasted by a cell in the System Information Block (SIB) [SBS12, 3GP12]. The value of a PCI can range between 0 and 503, and is used as a cell identifier for UE handover procedures. To have a successful handover, the PCI allocation procedure has to make sure that the network is PCI collision and confusion free. The first property guarantees that there is no cell in the network that has two or more neighbors with identical PCIs, whereas the second one ensures that there are no two neighboring cells receiving the same PCI. If those properties are not guaranteed, cells may not be able to properly handover UEs and, as a result, the handover performance in the network may decrease.

There are numerous reasons why an improper PCI allocation may occur. For instance, in cell outage management a common approach to close a coverage hole is to extend the coverage area of the surrounding cells. However, by doing so cells that were not assigned to be neighbors may start sharing a common coverage area [BRS+10]. As a result, PCI collision or confusion may occur. The same can happen when NEs are added to the network. As stated in [AFG+08], cell planning tools are typically used to predict the coverage of a new base station and generate the neighbor cell relation lists. Such tools, though, typically suffer from prediction errors due to imperfections of the available environment data, e.g., information about new buildings or streets. Consequently, wrong assumptions about the required neighbor relations can be made.

A PCI function implementation has been introduced in [BRS+10]. The solution itself makes use of minimum vertex coloring. The vertexes of the graph represent cells whereas edges connect cells that must not receive the same PCI. That is, for any two neighboring cells an edge is added to the graph which fulfills the collision free requirement. In addition, to make the assignment process confusion free, an edge is added for every two neighboring cells of second degree. The set of colors used by the algorithm represents the number of available PCIs.

8.2.2.2 ANR Function

The problem of having prediction errors about the network when using planning tools may not only affect the PCI assignment process. We may also have missing neighbor relations because of which handover performance may drop. For this reason, the ANR function has been developed [3GP16b, MBE11]. Its goal is to detect and establish neighbor relations by using active mobile terminals. As stated in [DJG+11], the ANR function mainly deals with the management of the Neighbor Relation Table (NRT), i.e., adding or removing neighbor relations. The NRT contains all the information a base station needs to know about its neighbors. Initially, the NRT can be configured by the operator, however, it may also be empty. The decision itself whether to add or remove an entry depends on the RRC signaling between a base station and the mobiles.

8.3 Summary

This chapter presented the simulation environment, also abbreviated as S3, that is used for the evaluation of the SON verification concept. It consist of a radio network simulator, a parser for the simulator, a set of self-optimization functions (MRO, TXP, and RET), a SON coordinator, and the verification mechanism. The network topology, the cell types, and all parameter settings have been introduced as well.

Furthermore, the used real data set has been described. It originates from an LTE network and consist of PM and CM data sets that have been exported in November and December 2013. The SON functions that were active during that time frame, ANR and PCI, are introduced as well. Both fall within the self-configuration class.

Chapter 9

Concept Implementation

In this chapter, the main topic of discussion is the implementation of the verification process which has been introduced and discussed in detail in Chapters 5, 6, and 7. In particular, a description of the components is given that implement the algorithms used for both CM and topology verification. Furthermore, an extensive overview of the libraries required to run the verification process is provided. Those are accompanied also with a description of the components that offer visualization and export capabilities which are mostly needed for the evaluation of the concept.

The chapter itself is divided into four parts. Section 9.1 is devoted to the verification process implementation. Section 9.2 describes the initialization procedure of the process. Section 9.3 introduces the external libraries that are required for the development of the introduced concept. Section 9.4 summarizes the chapter and lists the contributions to the research objectives, as defined in the very beginning of this thesis.

9.1 Structure of the Verification Process

The verification process is comprised of two components: one realizing CM verification and another covering verification of topology changes. Both components are very similar in their implementation as well as share numerous interfaces and abstract classes between each other. The verification process is implemented in the Java programming language, version 8 [GJS+14]. Although the development process started back in 2013, the decision was made to use Java 8 as soon as possible. The main reason was the introduction a new set of features that were not previously available in version 7. The most important ones are lambda expressions, `Optional` types, default and static methods in interfaces, the integration of the stream Application Programming Interface (API) into the collections API, and functional interfaces. Those changes simplified the development and improved the code structure and readability. Nonetheless, additional packages were used as well, which are discussed in detail in Section 9.3.

The implementation itself can be split into a set of main components (cf. Section 9.1.1), recorders (cf. Section 9.1.2), and set of utilities (cf. Section 9.1.3). Furthermore, the

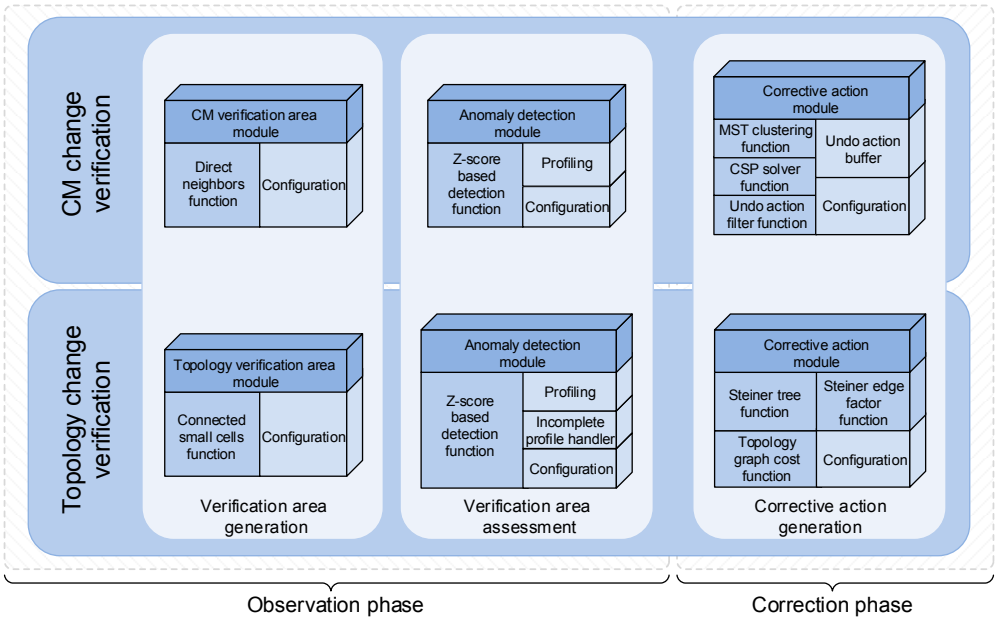


Figure 9.1: Overview of the verification process components

implementation offers Graphical User Interface (GUI) components that provide visualization capabilities during an experiment, as well as an API that allows the initialization and usage of the verification process itself. Sections 9.1.4 and 9.2.1 describe those in detail. Note that all components are depicted in Figure 9.1.

9.1.1 Main Components

In total, there are three main components that are implementing the core of the verification process. First of all, we have functions which provide the implementation of the core mechanisms, e.g., the MST-based clustering approach and the method that identifies soft collisions. Second, there are modules which initialize the functions as well as handle their initial setup. In most cases, they make use of `conf` files which are read during the initialization phase. Third, there are type classes that define verification specific objects, e.g., verification areas and cell clusters. In the upcoming paragraphs, those components are going to be described in detail.

9.1.1.1 Functions

The implementation of the verification process defines function subcategories:

- Verification area functions
- Anomaly detection functions
- Corrective action functions

Within the first class fall functions that handle the fragmentation of the verification scope (cf. Definition 5.1) as well as the formation of verification areas (cf. Definition 5.3). They follow the directive described in Section 5.2, i.e., there are functions that implement the area selection when facing topology changes and such that from areas in the case of CM changes.

The second class represents functions that implement the verification assessment phase, i.e., methods that detect anomalies as well as handle the cell profiling, as given by Definitions 3.10 and 3.11. The latter one includes two operations. On the one hand, it is possible to export the currently recorded KPIs to a profile. On the other hand, already recorded profiles can be read in before even starting a test run. The outcome of a such function is a set of potentially anomalous verification areas.

Within the third category fall functions that process anomalous verification areas and generate undo actions, as introduced in Section 5.4. Thereby, they identify verification collisions, eliminate weak ones, find soft collisions, and assemble a corrective action plan that has the properties of being collision-free and gain-aware. Note that an introduction to those terms is given in Section 5.5, whereas the solution itself is presented in Sections 6.3 to 6.6.

In addition, for the purposes of topology verification there is a function that generates corrective actions as presented in Section 7.4.

9.1.1.2 Modules

Modules initialize verification area, anomaly detection, and corrective action functions as well as handle their initial configuration. Usually, there is a separate module for each function subcategory. Listing 9.1 shows a simplified code fragment that originates from the corrective action module. Here, the module's implementation allows the usage of several corrective action functions which are selected over a configuration parameter. The latter one is defined by the `s3.verificator.functionType` string in the corresponding `conf` file. The parameter itself is loaded by `ConfigFactory`, the configuration library for Java Virtual Machine (JVM) languages. Note that Section 9.3.3.4 provides more information about this package. Furthermore, it should be noted that the `@Getter` annotation is offered by the lombok package, which is discussed in more detail in Section 9.3.4.1. It provides a getter method with protected access level, i.e., only visible within the same package.

Furthermore, in the case of CM verification there is an additional module that buffers the corrective action plan. The module itself is referred to as an undo action buffer which temporarily stores undo actions that are being suppressed because of verification collisions.

Finally, the classes implementing the CM and the topology verification process initialize the modules they require.

Listing 9.1: Code fragments of the corrective action module

```
public class CorrectiveActionModuleImpl {
    @Getter(AccessLevel.PROTECTED)
    private final CorrectiveActionFunction correctiveActionFunction;
    private final String functionType =
        ConfigFactory.load().getString("s3.verificator.functionType");

    public CorrectiveActionModuleImpl() {
        switch (functionType) {
        case "MstClusteringFunction":
            correctiveActionFunction = new MstClusteringFunctionImpl();
            break;
        case "SoftCspFunction":
            correctiveActionFunction = new SoftCspFunctionImpl();
            break;
        default:
            throw new RuntimeException("Unknown function");
        }
    ...
}
```

9.1.1.3 Types

The implementation of the verification process distinguishes between there three main type classes. First of all, we have objects that are used by both the CM and the topology verification implementation. Among those are the anomaly level vector and the cell area. The latter one is the parent class of those implementing verification areas (cf. Section 5.2).

Second, there are objects that are specified only for the purpose of CM verification. In particular, we have an object that represents a verification area that is formed around a reconfigured cell, as well as objects that represent the CVSI (cf. Section 6.2), and a cell cluster formed after applying the MST clustering approach (cf. Section 6.3). In addition, the outcome of the CM verification process is summarized by an undo result object that contains the set of cells whose configuration is marked for a rollback.

Third, type classes are defined that are considered only by the implementation of topology verification process. Concretely, we have an object that defines the verification area that is formed based on topology changes, as well as an object that recaps the outcome of the Steiner tree algorithm (cf. Section 7.3). In addition, an object is specified that handles the topology verification graph edge factor, i.e., a variable that is used when computing the weights in the topology verification graph, as given by Definition 7.3.

9.1.2 Recorders

As the name suggests, recorders handle the import of data exported by the network. In general, there are three types of recorders: one accountable for storing the anomaly vectors, one responsible for the cell profiles and another for the network data itself, i.e., PM and CM exports. The following three sections are devoted to those components.

9.1.2.1 Profile Recorder

The profile recorder is responsible for storing the cell profiles by considering the exported cell KPIs. In particular, it allows the user to specify when exactly the network training phase begins, when it ends, as well as where to save the cell profile data. The latter one is represented in `json` format. The library that handles the import, parsing, and export of `json` files is gson, also known as a Java serialization and deserialization library that converts Java Objects into the `json` format and back. An overview of this library is given in Section 9.3.3.1.

9.1.2.2 Anomaly Level Vector Recorder

The anomaly level recorder keeps track on the current CKPI / TKPI anomaly level vectors that are computed by the anomaly detection function. Note that the latter one is discussed in Section 9.1.1.1. It stores those vectors in a map as well as exports them in `csv` format for evaluation purposes.

9.1.2.3 Network Data Recorder

The network data recorder is responsible for buffering PM and CM data that are exported by the network. In the terms of the simulation environment, it gets triggered after completion of a simulation round. It should be kept in mind that the term simulation round is extensively discussed in Section 8.1.1.1. Furthermore, the recorder handles the export of those items, i.e., for evaluation purposes it saves them as `csv` files.

9.1.3 Utilities

The implementation of the verification process defines two utility types. On the one hand, there are such that provide extra mathematical functions and operations. On the other hand, there are utilities that offer functionalities for interpreting the network topology. Sections 9.1.3.1 and 9.1.3.2 are devoted to those topics.

9.1.3.1 Mathematical Utilities

Here, we can further distinguish between two utility types. The first one offers implementations of the core algorithms used by the verification process, that is:

- Minimum graph coloring by using backtracking [Wei15].

- The Steiner tree algorithm by Kou, Markowsky and Berman [KMB81].
- Class that posts and solves a Constraint Satisfaction Problem (CSP).
- Policies and ratings that further specify a CSP.
- Class handling the exponential smoothing of input data.

The reason why the code is refactored like that is to improve both extensibility and maintainability. Moreover, some of those classes are instantiated by multiple verification components. The most obvious example is the class implementing exponential smoothing capabilities which is required by both the CM and topology verification process.

The second type provides static methods that can be called by any class being part of the verification process. In particular, we have:

- Static methods that perform mathematical operations on the input data.
- Static methods that transform and manipulate JGraphT objects.

The JGraphT library itself is explained in Section 9.3.1.1, however, it is worth mentioning why there is a need to further define methods that manipulate graph objects. The library offers various graph optimization algorithms as well as operations for accessing and changing the properties of graphs. Nonetheless, there are use cases that are not covered by the library, e.g., for the visualization of the Steiner tree-based verification approach the export of `dot` files is required. Those files are fed into Graphviz [EGK+04] which is a collection of utilities that renders graphs based on the DOT language syntax. Note that the latter one is a plain text graph description language, as described in [GKN15].

Another use case is the need to transform one collection type into another. Let us give an example by listing the following method:

Listing 9.2: Getting node cost table out of the topology verification graph

```
Table<T, T, Double> getSteinerCostTable(final WeightedGraph<T,
    DefaultWeightedEdge> topVerifGraph){
    ...
}
```

The topology verification graph is used as input by the `getSteinerCostTable()` method. It returns a guava table (cf. Section 9.3.2.1) that stores the costs between every pair of `<T>` objects, based on which the Steiner tree alogirthm forms the MST. Note that the algorithm itself is described in detail in Section 7.3 whereas the topology verification graph is given by Definition 7.3. Furthermore, it should be noticed that `<T>` is a generic type which in the terminology of the verification process corresponds to a cell.

9.1.3.2 Network Utilities

The methods provided by Java classes implementing network utilities can be grouped in the following way:

- Static methods that provide access to macro and small cells.
- Static methods that return neighbor relations between cells.

Both classes include constructs that simplify the access to network entities and resources. For example, returning the set of all small cells out of all network cells, or to the test whether a cell object is actually a small or a macro cell. Another example is the computation of neighbor relations only between macro cells. Such a method is particularly useful for the scenarios that do not utilize the Steiner tree-based verification approach.

9.1.4 GUI

The GUI developed to visualize the test run results is implemented by using JavaFX and FXML [Bie14]. JavaFX is a software platform that allows the creation of desktop applications that run on various devices. According to [Ora16], it is expected to become the standard GUI library for Java. FXML on the other side is an XML-based language that allows the separation of content representation and the application logic itself. Hence, there is a high flexibility degree when it comes to change the way of how the verification outcome is visualized.

Nevertheless, JavaFX was not the initial choice for developing the GUI elements. The very first choice was the Java Web Toolkit (JWt), which did not manage to provide the set of features that were required over time. Moreover, it has proven to be unstable in some cases as well as not well documented.

9.1.4.1 JavaFX and FXML Components

The implementation provides two pairs of JavaFX and FXML components, i.e., two different views. On the one hand, there is a pair that represents the data exported by the CM verification process. That is, the CKPIs as well as the CKPI anomaly levels, as introduced in Section 6.1. On the other hand, there is a view that enables the visualization of equivalent performance data generated by the topology verification process.

9.1.4.2 Plot Definitions

In order to visualize the performance of the network while an experiment is running, a custom line chart class is provided. It specifies not only the chart type, but also the data series type, the labeling, as well as how to handle nonexistent data points. The latter one is needed in the case of cells that do not possess a valid or up-to-date profile, as given by Definition 3.10.

9.2 Verification Process Usage

9.2.1 API

The API of the verification process provides access to the main building blocks of the implementation. In general, we can distinguish between three constructs that are offered to the user of the system:

- Classes allowing the access to the CM and the topology verification process.
- Classes representing types that should be also visible to classes outside the verification process implementation.
- Classes allowing the access to recorders.

The first category offers methods like the verification of a set of cells or reset all network states kept by the verification process. Within the second category fall objects that represent the performance state of the network, e.g., CKPI / TKPI anomaly level vectors. The third category is comprised of recorders that allow the export of those objects. As discussed in Section 9.1.2, they are required for evaluation and visualization purposes.

9.2.2 Package Structure

The package structure of the verification process implementation can represented as shown below. The are three main folders, namely `api`, `gui`, and `internal`. As a commonly used package structure, `api` and `gui` collect classes that offer the functionalities introduced in Sections 9.1.4 and 9.2.1. External components wishing to use the verification process should call the classes located in those two packages.

```
|-- api
|-- gui
|   |-- cm_verification
|   ‘-- topology_verification
‘-- internal
    |-- functions
    |-- impl_cm_verification
    |   |-- functions
    |   |-- modules
    |   |-- types
    |   ‘-- CmVerificationImpl.java
    |-- impl_topology_verification
    |   |-- functions
    |   |-- modules
    |   |-- types
    |   ‘-- TopologyVerificationImpl.java
```

```
|-- modules
|-- recorders
|-- types
‘-- util
```

The `internal` package contains the implementation of both the CM and the topology verification process. As the listing shows, each implementation has its own functions, modules, and types (cf. Section 9.1.1). Furthermore, there are such being generic which are located one directory level up. Typically, they are either abstract classes that act as parent classes of verification process components, or interfaces that define a common function, module, or type structure.

It should be noted, though, that this does not apply for the `util` package. As stated in Section 9.1.3, utilities are accessible by every verification process class which is also the reason why they have been put into this directory.

9.2.3 Initialization and Usage

Listing 9.3 shows a simplified initialization process of the simulation environment S3, as described in Section 8.1. At first, the initialization of the connector to the LTE network simulator is triggered. Then, in the `run()` method the CM verification function, the SON functions (cf. Section 8.1.2) as well as the GUI elements are initialized. Note that the simulation system sees the verification process as a SON function that performs changes to CM parameters.

Listing 9.3: Simplified initialization of the CM verification process

```
public class S3 {
    private final SimConnector simConn = new SimConnector();
    ...
    public void run(){
        final CmVerificationFunction verifFunction = new
            CmVerificationFunction();
        initDesiredFunctions();
        initGui();
        while (true) {
            simConn.readDataFromSim(); //blocks until PM data is exported
            final Set<Cell> degrCells = verifFunction.trigger();
            simConn.undoCells(degrCells); //rollback degraded cells
            triggerSonFunctions(Sets.difference(simmCon.getCells(),
                degrCells)); //run MRO, RET, and TXP for remaining cells
        }
    }
}
```

Afterwards, a `while` loop is called in which changes are verified that have been made during the previous simulation round. Those causing a degradation in performance are rolled back. Finally, the cells not being affected by an anomaly are passed to the method that triggers the MRO, RET, and TXP function.

9.3 Libraries and Packages

The implementation of the verification process requires a set of packages and libraries which provide additional functionalities and, in most cases, simplify the development of the components. In this section, those packages will be the major topic of discussion. First, the mathematical libraries are discussed in Section 9.3.1. Then, in Section 9.3.2, an overview to the additionally used collection types and methods is given. It is followed by Section 9.3.3 which is devoted to the tools used to provide logging capabilities. Finally, Section 9.3.4 presents the packages that extend the current set of Java annotations.

9.3.1 Mathematical Libraries

9.3.1.1 JGraphT

Both the CM and the topology verification process are heavily based on graph theory approaches. The Java language, however, does not provide mathematical graph theory objects and algorithms which is the main reason why an external library is considered. The library itself is called JGraphT [NC16] and is known to be focusing on data structures and algorithms. It should not be confused with JGraph [JGr16] which is a library that is devoted to GUI-based editing and rendering.

Concretely, all graphs mentioned in Chapters 5, 6, and 7 are implemented by using the objects provided by the JGraphT library. For example, the Euclidean graph formed during the MST clustering procedure (cf. Section 6.3) is a weighted graph whose nodes are cells, as indicated in Listing 9.4.

Listing 9.4: Code snippet that demonstrates the usage of JGraphT while performing cell clustering

```
...
final WeightedGraph<Cell, DefaultWeightedEdge> euclideanGraph =
    getEuclideanGraph(recorder,
    allCells, pmReader.getMostRecentGranPeriod());
final MinimumSpanningTree<Cell, DefaultWeightedEdge>
    euclideanMinSpanningTree = new KruskalMinimumSpanningTree<Cell,
    DefaultWeightedEdge>(
    euclideanGraph);
final Set<Cluster> clustersOfTree = getClusters(euclideanGraph,
    euclideanMinSpanningTree);
...
```

As the listing shows, the JGraphT library implements Kurskal's MST algorithm [Kru56, CLRS09] which is called numerous times by the verification process implementation. In particular, it is needed by the clustering procedure as mentioned above, as well as by the Steiner tree algorithm (cf. Section 7.3). The latter one also makes use of Dijkstra's algorithm [Dij59,CLRS09] which is provided by the very same library. It should be noted, though, that JGraphT does not provide an implementation of the Steiner tree algorithm by Kou, Markowsky and Berman [KMB81]. The algorithm itself is required when verifying topology changes.

9.3.1.2 Choco Solver

Choco is a free and open source Java library that is dedicated to constraint programming [PFL15]. It is required by the CM verification process when generating a corrective action plan which has to be collision-free and gain-aware as given by Definitions 3.5 and 3.6. In particular, the approaches described in Section 6.5 and 6.6 make use of the Choco solver. Listing 9.5 shows the code snippet that implements the posting of the verification collision constraint.

Listing 9.5: Code snippet that shows the posting of hard constraints while generating the corrective action plan

```
    private final Map<IntVar, Set<IntVar>> hardVarMap = new HashMap<>();
    private final Solver solver = new Solver();
    ...
    private void postHardConstraints() {
        for (final IntVar var1 : hardVarMap.keySet()) {
            final Set<IntVar> variables = hardVarMap.get(var1);
            verify(!variables.isEmpty());
            for (final IntVar var2 : variables) {
                final Constraint verifCollision =
                    IntConstraintFactory.arithm(var1, "!=", var2);
                solver.post(verifCollision);
            }
        }
    }
```

First of all, the solver and the `hardVarMap` map are initialized. The purpose of the latter one is to buffer the verification collisions, i.e., a mapping of a cell to all cells that are in collision with it. Note that the cells are of `IntVar` type which may accept integer values between 1 and the maximum number of correction window slots, as presented in detail in Section 6.5.

The `postHardConstraints()` method shows the actual implementation of defining a collision and passing it to the solver. It goes between every pair of variables and requires

them to be unequal, i.e., the undo actions for cells associated with those variables must not be allocated to the same correction window time slot.

9.3.1.3 Apache Commons Math

The Apache commons mathematics library offers self-contained and lightweight mathematics and statistics components for addressing problems that cannot be solved by solely using the Java language [Fou16]. During the development process, the library has proven itself in being stable and well documented.

The most common methods that are used by the verification process implementation are those that normalize an array of values, as well as methods provided by the `FastMath` class. According to its developers, `FastMath` is a faster and more accurate alternative to the methods offered by well-known classes like `Math` and `SctrictMath`. This applies especially for large scale computations.

9.3.2 Extra Collection Types, Methods, and Conditions

9.3.2.1 Google Guava

Guava is an open-source collection of common libraries that have been developed by Google [Goo15]. It provides a variety of new collection types, caching capabilities, new string processing mechanisms, as well as utilities that, for example, add preconditions to methods or verify their outcome. Of course, the implementation of the verification process does not require all of them, but only a portion. Hence, in the upcoming paragraphs an overview to this subset of Guava features will be given.

First and foremost, the verification code extensively uses Tables, as previously shown in Listing 9.2. They are defined as `Table<R, C, V>`, where `R` specifies the row type, `C` the column type, and `V` the value type. There are several table implementations that are provided by Guava, however, the hash-based table is the one that is used in the verification process code. A Guava hash-based table can be seen as a nested hash map, i.e., `HashMap<R, HashMap<C, V>>`, that provides several useful methods like getting the set of all table cells or the entries associated with a given row.

Second, the verification code makes quite often use of immutable collections. Let us consider the following constructor (cf. Listing 9.6) which highlights the necessity of such collections. As stated in Section 5.2, a verification area consist of the reconfigured cell, also known as the target cell, and a set of cells surrounding that cell, which is commonly referred to as the target extension set. When initializing verification areas, the advantage of immutable collections becomes obvious. Similarly to `targetCell`, the elements of `extensionSet` must be unchangeable, i.e., even if we have a reference to the object we must not be able to remove or add any set items. Otherwise, undesired modifications may occur which may alter the outcome of the verification process.

Listing 9.6: Code snippet that demonstrates the usage of immutable collections

```
...
private final Cell targetCell;
private final Set<Cell> extensionSet;

public VerificationArea(final Cell targetCell, final Set<Cell>
    extensionSet) {
    this.targetCell = targetCell;
    this.extensionSet = ImmutableSet.copyOf(extensionSet);
}
...
```

Theoretically, `unmodifiable` methods provided Java Collections can be used here. However, according to the Guava developers those methods are unwieldy and inefficient. In addition, they are not really unchangeable since collections are immutable only if no one is holding a reference to the object.

Third, Guava's conditional failures are used in various places in the verification process code. Let us recall the example shown in Listing 9.5, which demonstrates how verification collisions are defined as constraints and how they are passed to the Choco solver. In particular, let us put the focus on the `verify()` method. In general, it accepts a `boolean` expression and throws a `VerifyException` if the expression is false. In addition, custom messages can be defined.

In a similar manner, Guava's preconditions are called at several places in the verification code. Concretely, it is quite often the case that `checkArgument()` is called at the very beginning of methods, e.g., such performing mathematical computations. The method itself takes the same arguments as `verify()` does, but throws a `IllegalArgumentException` if the passed expression is not true.

As it can be seen, those two methods are very much alike and even resemble `assert` methods in Java. However, there are differences between asserts and Guava's conditional failures. First of all, asserts are not turned on by default. In order to enable them, the user has to add the `-ea` argument when starting the JVM. Second, asserts have been developed to support the development process by providing sanity checks until the code gets finished. They are not supposed to consistently check, for example, if a KPI is currently zero. In addition, enabling assertions means that such checks are activated for all classes, i.e., also those provided by extra libraries. As a result, if we get an assertion error it may be a time consuming task until we finally identify the source of the exception.

9.3.2.2 Apache Commons Lang

The implementation of the verification process depends on Apache commons lang [Fou16]. The library itself is most notably known for its String manipulation methods, several helper utilities, as well as numerical methods. The verification process makes use of the

`ArrayUtils` class which provides convenient methods for manipulating arrays. It should be noted, though, that arrays are used in the verification code only for mathematical computations. For example, methods provided by `StatUtils` often require primitive data types as arguments. As a result, the verification code frequently uses the `toPrimitive()` method offered by the array utilities.

9.3.3 Logging and Configuration

9.3.3.1 Google Gson

Gson is an open-source Java library that converts Java objects into their `json` format. It is currently being developed by Google [Goo16]. The goals of the library is to provide convenient and simple `toJson()` and `fromJson()` methods, to allow custom representations for objects as well as offer support for arbitrarily complex objects.

The implementation of the verification process requires a parser for `json` files while updating or storing cell profiles (cf. Definition 3.10). Such files are imported at the beginning of a test run.

9.3.3.2 Apache Commons IO/CSV

The Apache commons IO library provides a set of utilities that extend the IO functionalities provided by the Java language [Fou16]. The verification process implementation makes extensive use of file manipulation utilities, e.g., when initializing a test run scenario. Furthermore, it uses the Apache commons CSV library which implements useful methods for handling `csv` files. Such files usually contain statistics that have been collected during a test run and are used later for evaluation purposes.

9.3.3.3 Apache Log4j

Apache Log4j is a logging and tracing library for the Java language [Fou15]. The implementation of the verification process makes use of this library at several places. Furthermore, it defines different logging levels, e.g., such listing only error logs and traces. The logs are exported to files or in the terminal.

9.3.3.4 Configuration Library

As mentioned in Section 9.1.1.2, the configuration of the verification process is handled by the configuration library for JVM languages [Lig16], developed by Lightbend (formerly Typesafe). The library itself stores the configuration parameters in the following files:

- `application.conf`
- `application.json`
- `application.properties`

- `reference.conf`

The order of the listing also defines the priority when reading in a configuration. Hence, default configurations are specified in the `application.conf` whereas custom changes can be defined in the remaining configuration files. Specifying a custom parameter overwrites the default one. If we recall the example from Listing 9.1, we would get the following `conf` file:

Listing 9.7: Overview of the default verification process configuration file

```
s3.verificator{
    ...
    # Corrective action configuration
    functionType="MstClusteringFunction"

    # MST clustering configuration
    factorEdgeRemoval=1.5
    ...
}
```

9.3.4 Annotations

9.3.4.1 Lombok

Over the last years, the Java programming language has been criticized for the volume of repeated code that gets generated in most development projects [Kim16]. In most cases such code originates from getter and setter methods, that provide access to class variables, constructors, `toString()`, `equals()`, as well as `hashCode()` methods. Especially the latter two increase the chances of bugs in the code. For example, every time we add new class variables we would need to update the `equals()` and `hashCode()` methods appropriately.

The Lombok library is designed to overcome such issues by providing a set of annotations that ease the generation and update of such methods [vdHEZ+16]. Hence, Lombok simplifies the development process and improves code readability. Let us give an example that demonstrates that ability. Listing 9.8 shows parts of the `VerificationArea` class that makes use of Lombok annotations. At the very beginning, we can see the `@Getter` annotation which generates a getter method for every class variable. It is followed by `@EqualsAndHashCode` which, as the name suggests, automatically provides an `equals()` and `hashCode()` method. Note that in this particular example the variables, based on which those methods are generated, are explicitly defined by the `(of = {...})` statement. Of course, it is also possible to omit this specification which will result in the inclusion of all listed class variables.

Finally, we have the `@ToString` annotation that provides a `toString()` method. Now, if we update the `VerificationArea` class we are not obligated to update those methods since they are automatically regenerated by Lombok.

Listing 9.8: Code snippet that shows the usage of lombok annotations in the verification area class

```
@Getter
@EqualsAndHashCode(of = { "baseCell", "extensionSet" })
@ToString
public class VerificationArea {
    private final Cell baseCell;
    private final Set<Cell> extensionSet;
    ...
}
```

9.4 Summary

This chapter is devoted to the implementation of the concept of SON verification and the challenges that emerged during the development process. It discusses the structure of the verification code, including the main components, the different recorders, and the provided utilities. An overview of the GUI, and the offered API is presented as well. A major topic of discussion are the used external libraries and packages without which the implementation of the verification concept would not have been possible. The following contributions highlight their importance:

O1.2: Model and design a verification process that assesses CM changes as well as changes made in the network topology.
Q: How is the implementation of the verification process being structured?
A: The implementation of the verification process, i.e., CM and topology verification, is split into four parts: a set of main components, a set of recorders, a set of utilities, and elements that implement the GUI.
The main components provide the core features of the verification process and are further divided into functions, modules, and type classes. Functions implement the verification area selection, the anomaly detection, and the corrective action procedure. Modules initialize those functions as well as handle their configuration. Type classes define the objects that are used by both the CM and the topology verification process. Recorders handle the import of CM and PM data, as well as the storage of profiles and all anomaly level vectors.
Utilities implement the mathematical functions and operations, as well as offer functionalities for interpreting the network topology, e.g., providing access to neighbor relations.
The GUI elements are responsible for the visualization of the results of a test run.

Q: Which is the program language of choice?
A: The programming language of choice is Java, version 8. Although the development started back in 2013, the decision was made to use version 8 as soon as possible. The reason is the introduction of new features like lambda expressions, optional data types, default and static methods in interfaces, integration of the stream API into the collections API, and functional interfaces. Those features simplified the development process.

Q: Which libraries are used for mathematical computations?
A: In total, three additional open source libraries are required by the verification process implementation: JGraphT, Choco, and Apache Commons Math.
JGraphT provides graph data structures and algorithms, e.g., Kruskal's algorithm. Due to the fact that the verification concept is heavily based on graph theory, most verification components require this library.
Choco is a Java library that is dedicated to constraint programming. It is required to implement the verification collision solver as well the corrective plan generation phase.
Apache Commons Math provides self-contained and lightweight mathematics and statistics components that are not offered by the Java language. The library is called numerous times within the verification code.

Q: Were all necessary algorithms provided by those libraries?
A: No. Although those libraries are powerful in providing access to numerous algorithms and features, the implementation of additional algorithms is required. For instance, they do not provide an algorithm that solves the Steiner tree problem. Another example is the MST clustering approach. Algorithms implementing the formation of an MST are offered, however, the clustering feature is not.

Q: Which additional packages / libraries are used besides the afore-mentioned ones and why are they required?
A: The reason why additional packages are used is twofold: simplification of the development process and usage of features not available in Java 8. The list begins with Google Guava which provides a variety of new collection types, new string processing mechanisms, as well as mechanisms that allow the usage of code preconditions and verification (cf. Section 9.3.2.1). Furthermore, Google's Gson library is used to manipulate json files, mainly for cell profiling purposes (cf. Section 9.3.3.1).
The list also includes the other Apache Commons packages. In particular, IO/CSV (cf. Section 9.3.3.2), lang (cf. Section 9.3.2.2), as well as Apache log4j (cf. Section 9.3.3.3). They are used for file manipulation, array object manipulation, and logging purposes, respectively.

In addition, the configuration library for JVM languages is utilized (cf. Section 9.3.3.4) as well as the lombok package is used (cf. Section 9.3.4.1). The first one handles the configuration of the verification process whereas the second one simplifies the development process by providing additional Java annotations.

Chapter 10

Evaluation Methodology and Results

The evaluation presented in this chapter is split into two parts. On the one hand, the process of CM verification, as described in Chapter 6, is evaluated. It starts by estimating the verification limits of SON functions and showing the need for an external verification process (cf. Section 10.1). Then, the verification collision problem is studied, in particular, the consequences of neglecting collisions are observed (cf. Section 10.2). It is followed by an evaluation of the collision resolving strategy (cf. Section 10.3) as well as the ability to eliminate weak collisions (cf. Section 10.4). It should be noted that those two evaluations are based on both the real data and the simulation environment. The CM verification study concludes with the evaluation of the ability to handle fluctuating PM data (cf. Section 10.5).

On the other hand, the capabilities of the topology verification process, as introduced in Chapter 7, are evaluated. It studies the impact of topology changes on the verification process as well as the ability to provide an accurate corrective action. Section 10.6 is devoted to this topic.

Finally, the chapter concludes with a summary and an overview of the contributions to the research objectives, as introduced in Section 1.3.

Published work This chapter is based on already published papers. The complete list includes the following conference and journal papers: [TNSC14b], [TSC15], [TFSC15], [TATSC16a], [TAT16], [TATSC16b], as well as [TATSC16c]. Note that the figures have been reformatted in order to present the results in a uniform way. In contrast to the papers, the case studies in this chapter are explained in more detail.

10.1 Studying the Verification Limits of SON Functions

The very first topic that is discussed in the evaluation section is the estimation of the SON functions' limits to verify their own CM changes. As stated in Section 3.2.1, a function may require several steps until it finally manages to reach its optimization goal. In particular, an example with the CCO function was given. The function tries to determine

whether in the previous step it has optimally changed the antenna tilt or the transmission power. Should this be not the case, it may correct its decision by deploying a new CM change or rolling back the previous one. However, in the very same section it is also mentioned that a function is only able to partially perform verification on its own. It is mainly caused by the fact that functions have a limited view on the network, i.e., they are usually not interested in the changes made by other functions. Furthermore, SON functions monitor only the KPIs they are interested in, e.g., the CCO function observes only PM indicators that reflect the coverage and capacity of a cell.

Hence, several questions arise regarding the verification capabilities of SON functions. For instance, a function may wrongly assume that its change has caused a degradation in the network. Instead, the anomaly could have been induced by another function that has been recently active within the same area. Furthermore, those capabilities are becoming even more questionable in the case of SON functions that monitor a shared set of KPIs but change different CM parameters. In such a case, adjustments made by one of the functions may result in wrong statements about changes made by the others.

In this section, those issues will be the major topic of discussion, which are split into the following subtopics:

- ✓ Study the ability of SON functions to handle degradations in performance.
- ✓ Observe the capability of SON functions to verify their CM changes in a coordinated SON.
- ✓ Monitor the network behavior when a SON verification logic, that has a wider view on the network, is allowed to asses the functions' changes.
- ✓ Compare the setup that uses a verification logic with a setup that relies only on the verification capabilities of the deployed SON functions.

The study itself is carried out by using the simulation environment, as described in Section 8.1. In addition, it is split into two parts: Section 10.1.1 is devoted to the parameter choices whereas Section 10.1.2 describes the scenario and results.

10.1.1 Simulation Study Setup

10.1.1.1 Active SON Functions

The functions that are active during the test runs are MRO, RET, and TXP, as described in Section 8.1.2. The latter two are fitting perfectly well for performing the evaluation since they monitor the same set of KPIs, have the same objective, but modify different CM parameters. Hence, if TXP changes the transmission power in such a way that it negatively impacts the KPIs monitored by the RET function, the latter one may try to correct its past actions although it was not supposed to do so.

10.1.1.2 Verification Area Selection

The verification area that is formed around the reconfigured cell is identical to the impact area of the CM change, as defined by the SON coordination logic (cf. Section 8.1.3). The impact area itself is described in Section 2.3.2.3 and is discussed in [BRS11, Ban13].

10.1.1.3 CKPIs and Profiling

The CKPI vector $\mathbf{k}^{\perp} = (k_1^{\perp}, \ldots, k_n^{\perp})$ consist of the following two performance indicators:

- Handover Success Rate (HOSR)
- Channel Quality Indicator (CQI)

It should be noted that the CQI is computed as the weighted harmonic mean of the CQI channel efficiency (cf. Section 8.1.1.2).

An element $p_i^{\perp}$ of a CKPI profile vector $\mathbf{p}^{\perp} = (p_1^{\perp}, \ldots, p_n^{\perp})$ (cf. Section 5.3.2) is a sequence of t CKPI samples. They are recorded while the network is operating as expected. Let us denote those samples as $k_{i,1}^{\perp}, k_{i,2}^{\perp}, \cdots, k_{i,t}^{\perp}$, where $i \in [1; n]$. Those samples are collected during a so-called training phase, lasting 70 simulation rounds, i.e., $t = 70$. Note that the term simulation round is introduced in Section 8.1.1.1.

However, to determine the behavior of a cell we need the current CKPIs (cf. Section 6.1). Let us denote the current sample of the i^{th} CKPI as $k_{i,c}^{\perp}$. All collected samples, i.e., $k_{i,1}^{\perp}, k_{i,2}^{\perp}, \cdots, k_{i,t}^{\perp}, k_{i,c}^{\perp}$, are standardized by computing the z-score [FPP07] of each data point. The z-score of $k_{i,c}^{\perp}$ is the anomaly level of the i^{th} CKPI, also denoted as $a_i^{\perp}$. Each CKPI anomaly level is element of a CKPI anomaly vector $\mathbf{a}^{\perp} = (a_1^{\perp}, \ldots, a_n^{\perp})$ which gives us the cell state (cf. Definition 6.1).

Furthermore, the current observation (cf. Equation 6.5) is computed as the arithmetic average of the CKPI anomaly levels. The state update factor α is set to 1 (cf. Equation 6.4), i.e., past values are not taken into account when updating the cell anomaly level.

10.1.1.4 Compared Strategies

In total, two configurations are compared with each other during this study. The first one makes use of a pre-action SON coordinator (cf. Section 8.1.3) that manages CM change requests. The initial coordination priority is set as follows:

- $\text{Priority}_{RET} > \text{Priority}_{TXP} > \text{Priority}_{MRO}$

In addition, stateful SON functions verify their changes on their own, i.e., the verification process is disabled for this setup.

The second configuration involves the CM verification process which monitors the activity of the functions and generates undo actions if they cause an undesired network behavior. In order to make a fair comparison, the verification process has to request permission from the SON coordinator before deploying an undo action, i.e., from the coordinator's perspective it is seen as a SON function.

10.1.2 Scenario and Results

The experiment consist of 5 test runs, each lasting 18 simulation rounds. Each test run starts with the initial network setup. In addition, in simulation round 5 of every test run, the TXP function makes a decision to decrease the transmission power of two neighboring cells accompanied by a change of the step size (the transmission power delta).

Figure 10.1 visualizes the average cell anomaly level of the two cells and their direct neighbors. Note that the results include the 95% confidence intervals that are computed around the sample mean of 5 test runs. As shown, up to simulation round 5 the average cell anomaly level is almost at zero, i.e., the cells perform as expected. Figures 10.2(a) and 10.2(b), which represent the raw CQI and HOSR used to form the CKPI anomaly vector, evidence this observation. However, this immediately changes as soon as TXP makes the decision to change the transmission power. As the figures emphasize, the z-score-based cell anomaly level no longer takes values near zero. Thereby, the performance of all neighboring cells degrades which can be seen in the figures depicting the CQI and HOSR.

The question that arises here is how do the two configurations manage to compensate the TXP decision and optimize the network. On the one hand, the configuration that utilizes the CM verification process manages to immediately put the cells' performance within the expected range, i.e., anomaly level of zero (cf. Figure 10.1). On the other hand, the configuration that relies only on the functions' verification capabilities partially compensates the event. As presented, the average anomaly level slowly returns to the expected range, however, never reaches zero. The reason for that is the dynamic coordination mechanism which starts to reject a function's requests if has been frequently allowed to run, while another, higher prioritized function tries to make a change at the same time and within the same area. As a result, TXP is providing appropriate corrective actions,

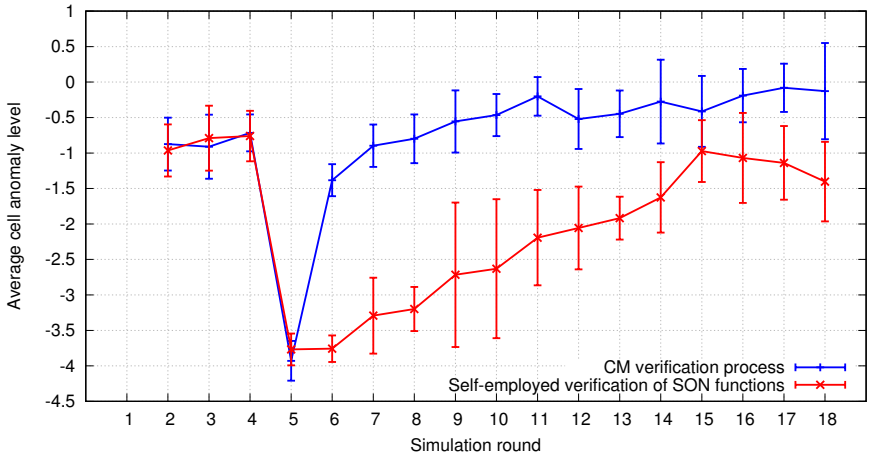
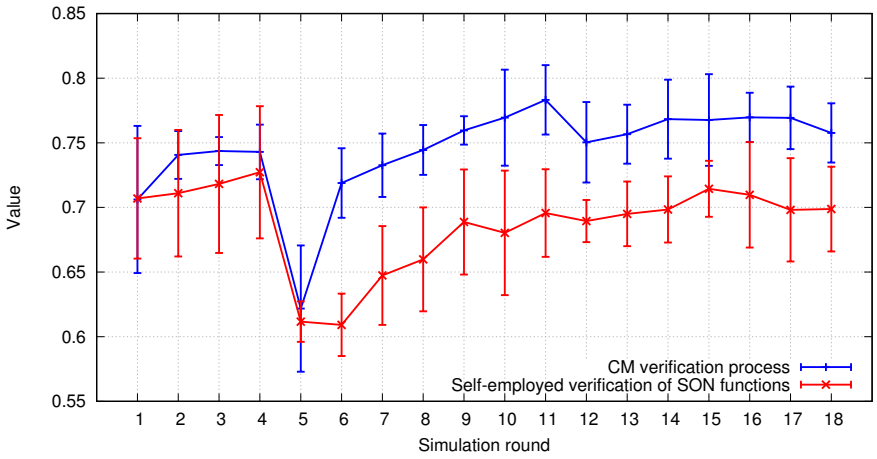
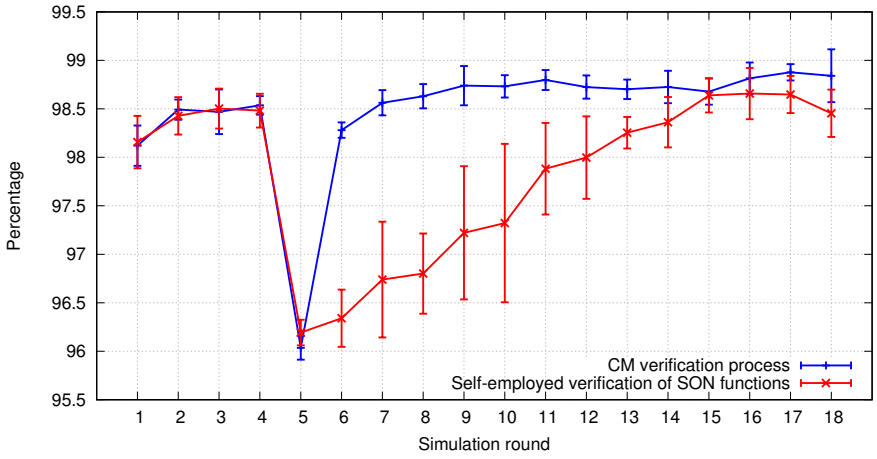


Figure 10.1: Studying the verification limits of SON functions: average cell anomaly level. The higher, the better. A value near zero indicates that the cells are performing as expected.



(a) The CQI KPI



(b) The HOSR KPI

Figure 10.2: Studying the verification limits of SON functions: raw KPI values

but is blocked by the coordinator. Moreover, the RET function is allowed to optimize the network, but its actions are not able to compensate the event. Those observations can be also seen in Figures 10.2(a) and 10.2(b).

10.2 Neglecting Verification Collisions

In this thesis, a lot of focus is put on the verification collision problem, as given by Definition 3.3. Basically, it is an uncertainty which prevents two undo actions from being simultaneously executed. The problem itself originates from the process of rolling back changes and the uncertainties that emerge when restoring older configurations. Section 3.2.2 is devoted to those particular issues.

The concept of SON verification addresses the verification collision problem. In particular, it provides solutions that eliminate weak collisions, resolve valid ones, as

well as find those that can be potentially violated. Sections 6.3 to 6.6 are devoted to those particular topics. However, before evaluating those methods, we first have to observe the consequences of neglecting verification collisions or, in general, uncertainties when rolling back configuration changes. Therefore, this section will be devoted to the following topics:

- ✓ The study of the verification collision problem in a coordinated SON.
- ✓ The comparison of the verification process with an approach that neglects the verification collision problem.
- ✓ Estimating the impact of neglecting verification collisions on the network performance.
- ✓ Estimating the limits of the compared approaches.

The study itself is simulation-based. The environment itself is described in Section 8.1. In the upcoming sections, the setup of the scenario is described (cf. Section 10.2.1), and the results of the study are outlined (cf. Section 10.2.2). Table 10.1 summarizes the selection of all relevant parameters.

10.2.1 Simulation Study Setup

10.2.1.1 Active SON Functions

In order to increase the likelihood of getting simultaneous CM change requests, the decision is made to use all available SON functions, i.e., MRO, RET, and TXP. An overview of those functions is given in Section 8.1.2.

10.2.1.2 Verification Area Selection

The verification area is formed by taking the reconfigured cell and its first degree neighbors. The impact area of the SON functions is set to be identical to the verification area.

10.2.1.3 CKPIs and Profiling

The CKPI vector $\mathbf{k}^{\perp} = (k_1^{\perp}, \ldots, k_n^{\perp})$ consists of two elements:

- Handover Success Rate (HOSR)
- Channel Quality Indicator (CQI)

The profiling procedure is identical to the one described in Section 10.1.1.3. The current observation and the cell anomaly level are computed as explained in the very same section. The cell anomaly level is considered as degraded in case it falls in the range $(\infty; -2.0]$.

Table 10.1: Study of neglecting verification collisions: parameter selection

Component	Parameter	Value
SON function collection	Set of SON functions	MRO, TXP, RET
SON coordinator	Initial priority setup	CM verification > RET > TXP > MRO
	Tokens	10 for each bucket
	Token change for Acks	-2 tokens
	Token change for Nacks	+1 token
Verification process	Training rounds	70
	Cell level degradation range	$(\infty; -2.0]$
	Cell level weights	0.5 for CQI and HOSR

10.2.1.4 Compared Strategies

In this study, two configurations are compared against each other. Both configurations utilize a SON coordinator and the CM verification process, i.e., the SON functions request permission from the coordinator before they change CM parameters. Deployed changes are later verified. Those configurations, however, differ in the way they handle undo action requests being in a verification conflict. The first one solely relies on the SON coordinator to resolve the conflicts. In particular, it makes use of the batch coordination concept with dynamic priorities, as defined in [RSB13]. Each SON function has an assigned bucket and dynamic priority. The bucket itself contains a number of tokens that are reduced every time a request is accepted and increased if it is rejected. In case of an empty bucket, the priority is set to the minimum value.

The second configuration makes use of the collision resolving approach as provided by the CM verification process. In addition, the coordinator considers the outcome of the collision resolving, i.e., it may deploy undo actions although their impact areas are overlapping.

10.2.2 Scenario and Results

10.2.2.1 Verifying a SON Optimization Process

The coverage of the system can be reduced and the its performance may degrade if the environment changes from the assumptions that were made when the network was planned and set up [HSS11]. Usually, such significant differences occur when buildings get demolished, base stations get inserted or removed, or when seasons change. As a result, inappropriate configuration decisions can get deployed which need to be assessed and eventually corrected by the verification process. To recreate such a scenario, the following 12 cells are selected: 1-3, 4-6, 16-18, and 22-24. The network coverage map and topology are depicted in Figure 8.3 and 8.4, respectively. Four of those cells, namely cells 2, 6, 16, and 24, are set up by using obsolete information about the environment.

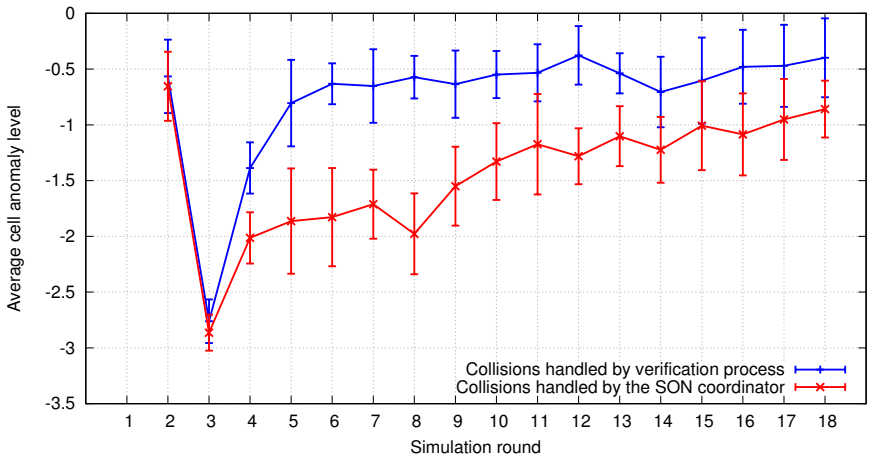


Figure 10.3: Study of neglecting verification collisions: average cell anomaly level. The higher, the better. A value near zero indicates that the cells are performing as expected.

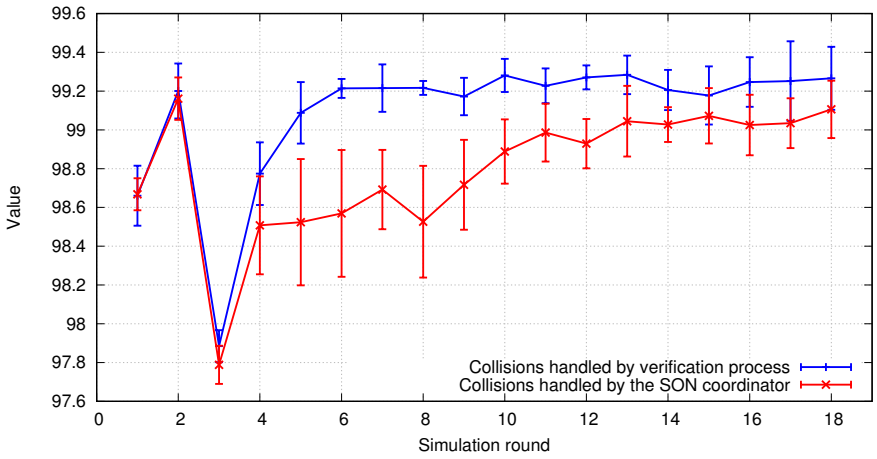
The experiment itself consist of 5 consecutive test runs, each lasting 18 rounds which gives us a simulated time of 27 hours. In simulation round 2 of every test run, obsolete transmission power and antenna tilt settings are applied. Then, the cell anomaly level is measured. Finally, the 95 % confidence interval is computed around the sample mean measured during those 5 test runs.

The results of this experiment are outlined in Figure 10.3. It shows the average cell anomaly level of all 12 cells. As the results outline, the configuration that makes use of the collision resolving approach (provided by the verification process) requires two simulation rounds to return the cell anomaly level to the expected range, i.e., near zero. The other configuration, that is, the one that is relying only on the SON coordinator to resolve the collisions, is not able to achieve such a result. Figures 10.4(a) and 10.4(b), which represent the HOSR and CQI, further evidence this observation.

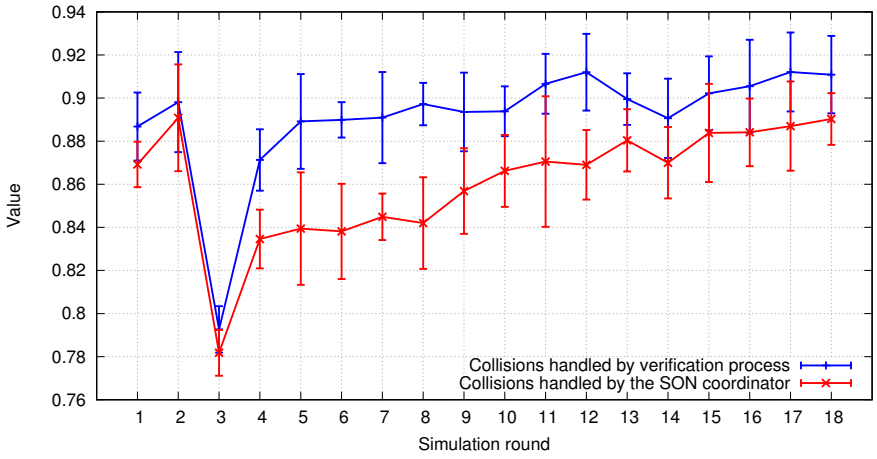
The cause of having such a difference in the cell anomaly level comes from how the two configurations handle verification collisions. The one that solely relies on the SON coordinator suppresses undo actions quite often due to overlapping verification areas. Furthermore, the dynamic coordination mechanism prevents the network to completely return the cell anomaly level as reported before round 2. The observations show that the SON coordinator shows a similar behavior, as discussed during the previous case study. It starts to reject the requests of a SON function if it has been frequently executed. As a result, RET as well as TXP are interrupted and cannot complete the optimization of the coverage of the affected cells.

10.2.2.2 Estimating the Limits

The study continues with the estimation of the limits of the compared approaches. In particular, the focus is put on studying the correlation between the number of undo actions



(a) The HOSR KPI



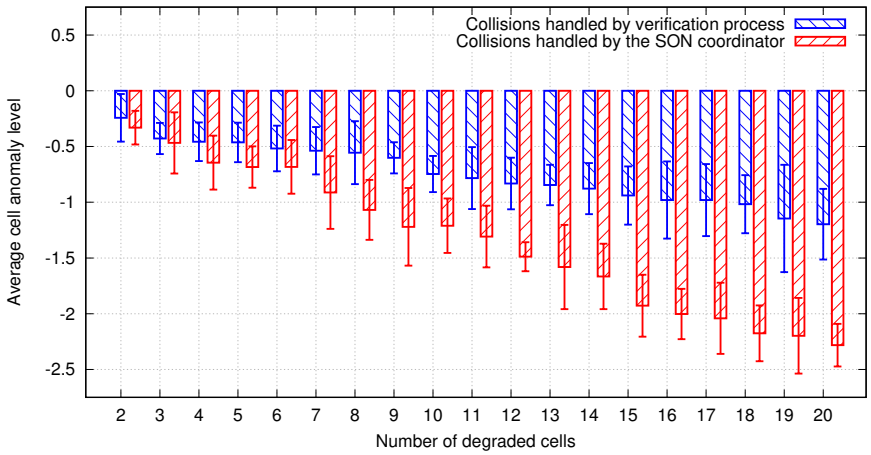
(b) The CQI KPI

Figure 10.4: Study of neglecting verification collisions: raw KPI values

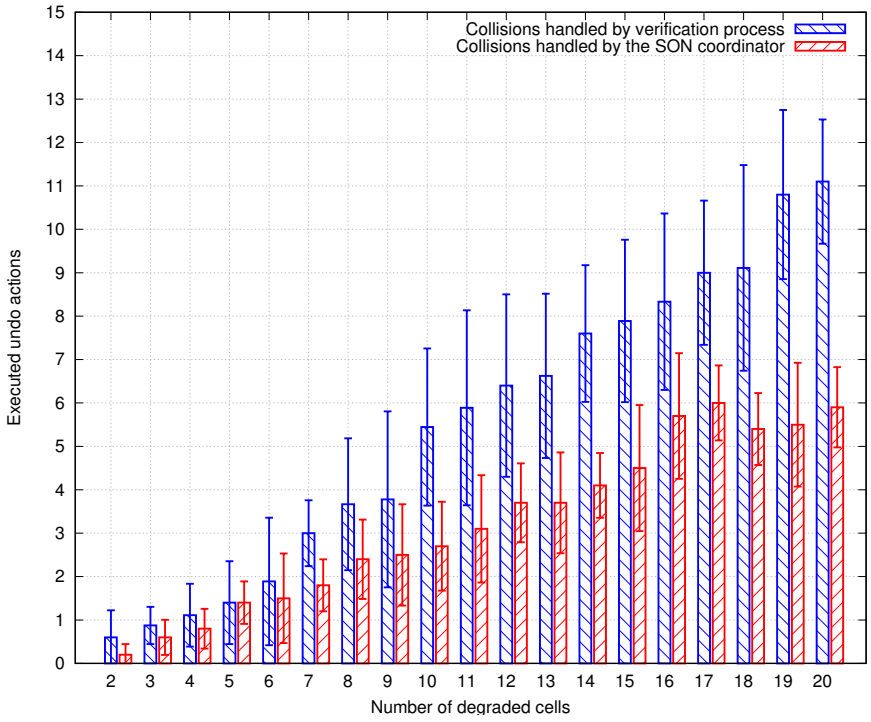
and the cell anomaly level as the number of degraded cells changes. The degradation itself is done by deploying a cell configuration that is unusual for the used network setup: a transmission power of 42 dBm and an antenna tilt of 3 degrees. Moreover, the number of degraded cells ranges between 2 and 20.

Each experiment consist of 9 test runs, each lasting 5 simulation rounds. When starting a new experiment, the total number of to be degraded cells is set. Then, the cells are randomly selected, where each cell can be chosen with equal likelihood. It is followed by inducing the degradation itself. At the end of the test run, the number of deployed undo actions is counted and the cell anomaly level is measured. The outcome of each experiment is measured by taking the arithmetic average of cell anomaly level and the number of executed undo actions over the 9 test runs. In addition, the 95% confidence intervals are computed.

The results of this study are visualized in Figure 10.5(a), which represents the cell



(a) Average cell anomaly level. The higher, the better. A value near zero indicates that the cells are performing as expected.



(b) Number of executed undo actions

Figure 10.5: Study of neglecting verification collisions: estimating the limits of the compared strategies

anomaly level for all 19 experiments. As shown, for a low number of degraded cells, i.e., up to six cells, the differences between the two configurations are minor. Nevertheless, this changes when at least seven cells (out of 32) degrade. As we increase the number of degraded cells, verification collisions simply start to occur more often. This can be seen in Figure 10.5(b) which shows the average number of executed undo actions for

each experiment. The configuration that relies only on the SON coordinator suppresses undo actions more often as the number of degraded cells increases. On the contrary, the configuration that makes use of the collision resolving strategy of the verification process provides a better cell anomaly level and lets more undo actions through. However, for a high number of degraded cells (20 out of 32) none of the configurations manages to completely restore the network performance, i.e., put the cell anomaly level near zero.

10.3 Solving of the Verification Collision Problem

The main topic of discussion in this section is the verification collision problem (cf. Definition 3.3), the generation of the corrective action plan (cf. Definition 3.4), the identification of an over-constrained plan (cf. Definition 3.7), and the search for soft verification collisions (cf. Definition 3.8). Thereby, the focus is on evaluating the approaches introduced in Sections 6.5 and 6.6.

The evaluation itself is split into two parts, described in Section 10.3.1 and 10.3.2, respectively. The first one is based on the real data set that is introduced in Section 8.2. It focuses on identifying verification collisions and analyzing their impact on the LTE network from which the data set originates. The second part is carried out on the simulation system that is described in Section 8.1. It is devoted to the analysis of the network performance after starting to process the blocks of the corrective action plan.

It should be noted that each study starts with a short motivation accompanied by the topics that are going to be discussed. Moreover, each study specifies the properties of the CM verification process, i.e., the selection of verification areas, the choice of CKPIs, the profiling procedure, and the choice of all functions required by the collision resolving procedure. Finally, each study concludes with the description of the scenario and the results.

10.3.1 Real Data Study

The motivation behind the real data study is to analyze the verification collision problem in a real network. In particular, the following topics are discussed:

- ✓ Analysis of the verification area selection.
- ✓ Detecting configuration changes that lead to anomalous KPI values.
- ✓ Generating a corrective action plan for the given setup and verification process parameters.
- ✓ Identifying soft verification collisions and readapting the corrective action plan that is generated by the CM verification process.

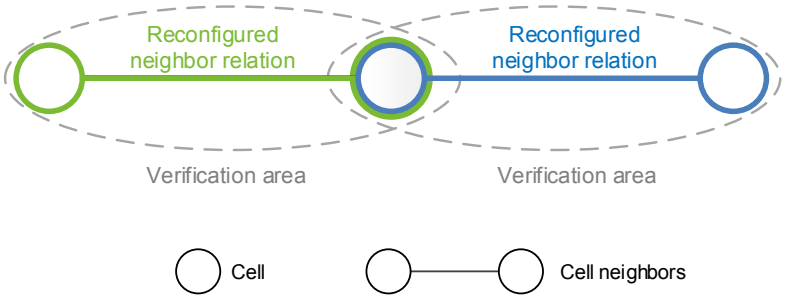


Figure 10.6: Neighbor relation changes and intersection of verification areas

10.3.1.1 Verification Area Selection

CM changes that have a high probability of causing verification collisions are reconfigurations of the neighbor relations, i.e., adding or removing cell neighborships. It is caused by the fact that the configuration of a neighbor relation always involves two cells. As a consequence, a verification area should at least include the cells of the reconfigured adjacency (cf. Figure 10.6). The reason for that comes from the consequences of improperly changing neighbor relations. On the one hand, PCI collisions may occur, i.e., two cells having the same PCI can become neighbors which may lead to reduced handover performance. Note that a detailed description of the PCI and the function assigning it is given in Section 8.2.2.1. On the other hand, PCI confusions, where a cell has two neighbors with the same PCI. In the same way, they can lead to a drop in performance. Figure 10.7 depicts a PCI confusion that leads to a verification collision. Initially, cell 1 has only one neighbor, namely cell 3. After adding the two new neighbor relations, cell 1 becomes a neighbor of two cells sharing the same PCI.

Therefore, in this study a verification area is set to comprise of the cells of the reconfigured adjacency. Furthermore, this decision is strengthened by the fact that during the three week observation period, 400 new cells have been added by the operator.

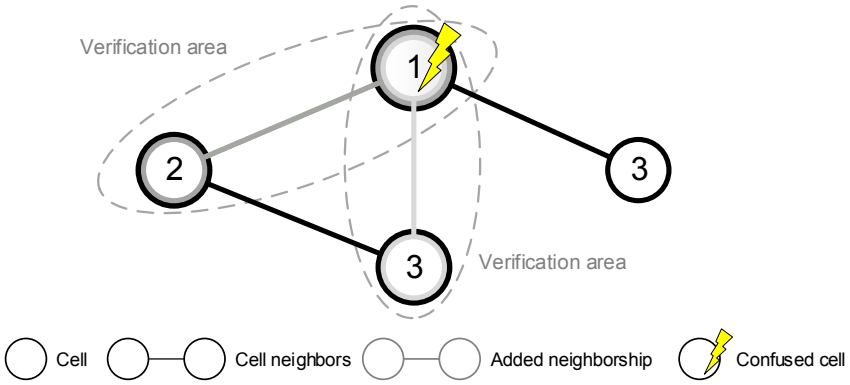


Figure 10.7: Correlation between the PCI confusion and verification problem

10.3.1.2 CKPIs and Profiling

The CKPI vector $\mathbf{k}^{\perp} = (k_1^{\perp}, \ldots, k_n^{\perp})$ is comprised of the following performance indicator:

- `EUTRAN_RLC_PDU_RETR_R_UL`: retransmission rate for RLC PDUs in uplink direction.

The profile that is used to detect anomalies is created by computing the z-score of PM data exported for that particular KPI. The exact way of recording the profiles resembles the example that has been presented in Section 6.1. In addition, a change is considered as anomalous if the z-score of the CKPI stays above 2.0 for at least two consecutive PM granularity periods, that is, two hours.

10.3.1.3 Scenario and Results

At first, let us observe the number of changes that have been made between the 25^{th} of November 2013 and the 17^{th} of December 2013, i.e., the observation time frame. Figure 10.8 outlines the number of adjacency managed objects that have been added for the given time interval. As it can be seen, numerous modifications took place on five out of seven days: the 28^{th} of November, as well as the 4^{th}, 5^{th}, 10^{th}, and 11^{th} of December.

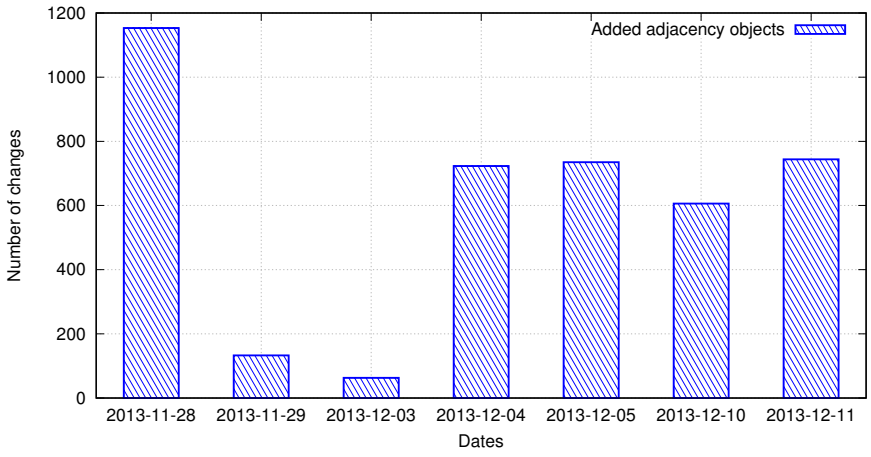
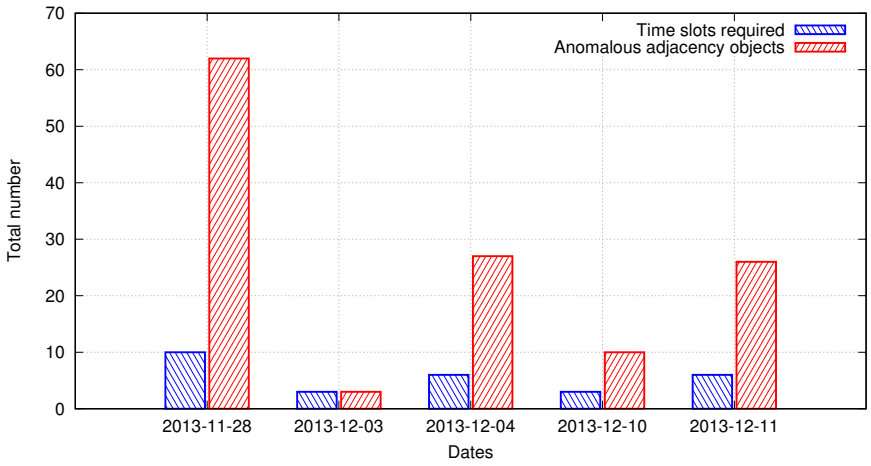


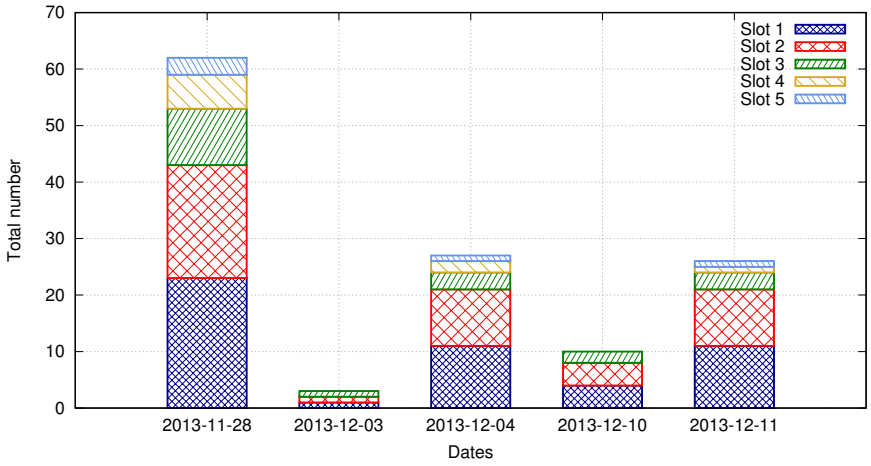
Figure 10.8: Number of added cell adjacency objects

As a next step, the focus is on detecting CM changes followed by abnormal CKPI values. Figure 10.9(a) visualizes the outcome of this observation. It shows the number of anomalous objects and time slots, each with the length of one hour, that are required if we attempt to undo the changes. The worst performance outcome is reported on the 28^{th} of November: 62 anomalous objects for which the verification process requires 10 correction window time slots.

The need of 10 slots, though, means that we have ten sets of undo actions which in the worst case scenario have to be sequentially deployed. Even if we manage to quickly process those sets and deploy the suggested changes, we would still depend on the PM



(a) Anomalous objects and number of correction window time slots required for the deployment of all undo actions



(b) Undo action distribution for a correction window of five slots

Figure 10.9: Results of the real data verification collision study

granularity. As stated in Section 3.3.2, the visibility of the changes' impact on the PM data is a major factor that has be taken into consideration. The LTE network that is evaluated here has a PM granularity of one hour which means that the only way of making the impact of the undo actions visible is to deploy each set immediately after PM data gets exported.

Therefore, the decision is made to halve the maximum number of required time slots and observe the impact on the corrective action plan. As a consequence, we get an over-constrained verification collision problem: for the 28^{th} of November, the 4^{th} and the 11^{th} of December. The resulting undo action distribution is visualized in Figure 10.9(b). As depicted, we get a new corrective action plan that allocates most of the undo actions to the first two slots.

10.3.2 Simulation Study

The idea of the simulation study is to evaluate the parts of the collision resolving procedure that were not discussed in real data study. Hence, it primarily focuses on the following subjects:

- ✓ Resolving verification collisions by varying the number of degraded cells.
- ✓ Observing the impact of an over-constrained corrective action plan on the overall network performance.
- ✓ Monitoring the number of executed undo actions as well as the cell anomaly level after processing the blocks of the corrective action plan.
- ✓ Study the interaction with other SON functions.
- ✓ Comparing the approach with other collision resolving strategies.

10.3.2.1 Active SON Function

One of the major differences between the simulation and the real data study is the presence of SON functions that are performing changes while the corrective action plan gets processed. In particular, the RET and TXP functions are allowed to adapt the antenna tilt and transmission power, as described in Section 8.1.2. As a result, we may have function actions that interfere with the verification process.

10.3.2.2 Verification Area Selection

A verification area is formed by selecting the reconfigured cell as well as all of its first degree neighbors. The motivation for selecting the area in such a way is given by the CM change type. Performing changes to the physical cell borders affects not only the reconfigured cell, but also cells that have a direct neighborship with the reconfigured one.

10.3.2.3 CKPIs and Profiling

The CKPIs vector $\mathbf{k}^{\perp} = (k_1^{\perp}, \ldots, k_n^{\perp})$ consists of the following indicators:

- Handover Success Rate (HOSR)
- Channel Quality Indicator (CQI)

The profiling procedure is identical to the one described in Section 10.1.1.3. That is, the CKPI anomaly level is computed as the z-score of the current CKPI value. The used profile is recorded before carrying out the experiments, in particular, a phase of 70 simulation rounds during which the network performs as expected.

10.3.2.4 Cell and Verification Area Assessment

Because of the nature of the z-score, a data point is considered as an outlier when the z-score value is either below -2.0 or above 2.0. In other words, it is two standard deviations away from the expected mean. However, due to the fact that only success KPI are selected as CKPIs, the HOSR and CQI are considered as degraded when their z-score falls in the range $(-\infty; -2.0]$. Note that the success KPI terminology is introduced in Section 5.3.1.

Furthermore, the current observation ψ (cf. Equation 6.5) is computed as the arithmetic average of all anomaly level vector elements, i.e., the average of the HOSR and CQI z-score. In addition, the state update factor α (cf. Section 6.2) is set to 1. As a result, the cell anomaly level ϑ is the arithmetic average of the CKPI anomaly levels.

A verification area is considered as anomalous when the average cell anomaly level falls below -2.0.

10.3.2.5 Priority Functions

Functions ρ (Equation 6.11) and $\tilde{\rho}$ (Equation 6.14) compute the priority by taking the average cell anomaly level of the considered cells.

10.3.2.6 Compared Strategies

The idea of this study is to observe the network performance when we start deploying undo actions. Moreover, it is of high interest to monitor the cell anomaly level, as well as how it correlates with the number of executed undo actions. However, the issue of grouping actions based on constraints, and finding the appropriate deployment order can be also represented as a scheduling problem. Approaches that are widely used to solve such scheduling problems are based on graph coloring [Lei79, Mar04]. The actions are represented in a graph and are assigned to different groups depending on the color they receive. Graph coloring approaches are also of particular interest since they are constraint satisfaction approaches that typically do not perform any type of constraint softening.

Therefore, a minimum vertex coloring approach is used for comparison. It schedules the undo actions based on the color frequency, i.e., the actions assigned to the largest group are deployed at first. Processing the groups based on their size creates also a potential for rolling back changes that did not harm network performance.

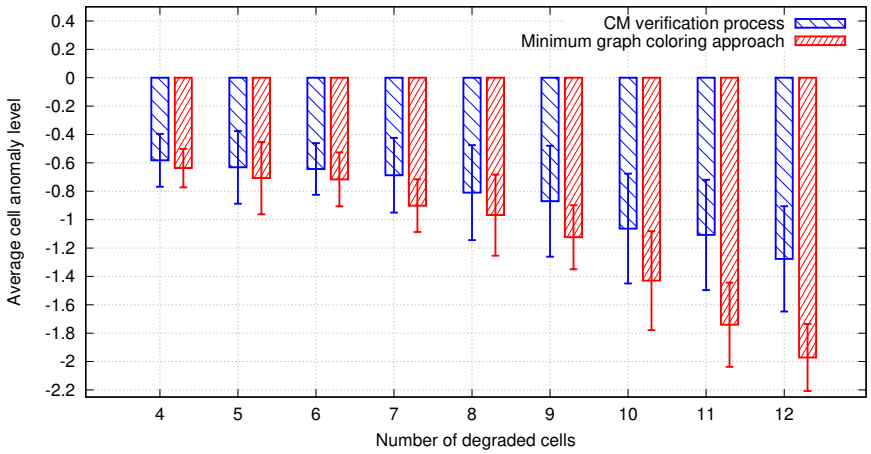
10.3.2.7 Scenario and Results

In total, 9 consecutive experiments are carried out. Each experiment is comprised of 13 test runs. Every test run lasts 4 simulation rounds, during which cells (ranging between 4 and 12) are selected for degradation. The exact number of cells is chosen before starting an experiment. The selection is made by adding all cell identifiers to a list, permuting the list, where all permutations occur with equal likelihood, and selecting the

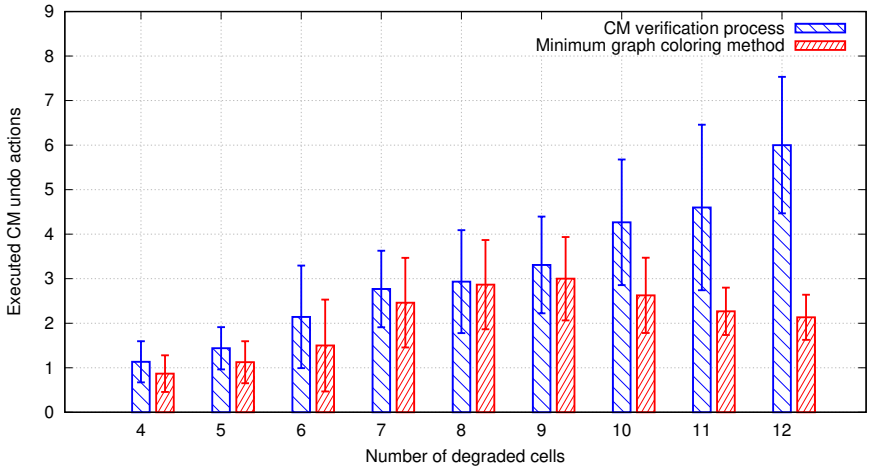
first n items. The degradation is carried out by deploying an untypical configuration for the given network scenario: an antenna tilt of 3 degrees and a transmission power of 42 dBm. Furthermore, the number of available time slots τ is set to 2 due to the size of the simulated network (cf. Section 8.1.1.5). The RET and TXP functions run in the background as well.

The measurement itself is carried out as follows. During the first simulation round of a test run degradations are induced. During the second and third round, the corrective action plan is processed and the suggested undo actions are deployed. The result itself is computed by taking the arithmetic average of the cell anomaly levels of all 32 cells at the end of the fourth simulation round.

Figure 10.10(a) outlines the results. Note that the results show the 95 % confidence



(a) Average cell anomaly level. The higher, the better. A value near zero indicates that the cells are performing as expected.



(b) Number of excuted undo actions

Figure 10.10: Results of the verification collision simulation study

intervals that are computed around the sample mean for each experiment. During all experiments, the collision resolving method of the CM verification process is able to provide a better cell anomaly level compared the minimum graph coloring approach. In the case of 4, 5 and 6 degraded cells, there is almost no difference in the cell anomaly level. The minimum graph coloring approach performs slightly worse due to ongoing SON function changes. Due to its strategy of deploying the largest set of undo actions at first, it forces necessary changes to be rolled back.

Moreover, we see that in case of 9 or more degraded cells, the performance of the minimum graph coloring rapidly drops. It is caused by the selected maximum of two time slots, which becomes the limiting factor. This trend can be seen in Figure 10.10(b) which gives us the actual number of deployed undo actions. The observations show that verification collisions start to appear more frequently as we increase the number of degraded cells, which also leads to a decrease of executed undo actions.

Another reason for this decrease is the activity of the SON functions in the regions where undo actions have been delayed due to collisions. At the time they saw the degradation, RET and TXP became active and triggered CM changes that required verification. Those changes were rolled back by both configurations, which caused the cell anomaly level not to return to a value near zero. Despite those activities, the CM verification process manages to approximately halve the cell anomaly level compared to the graph coloring approach.

10.4 Elimination of Weak Verification Collisions

In this section, the main topic of discussion are verification collisions (cf. Definition 3.3), in particular, the identification and elimination of such being weak, as given by Definition 3.9. The approach that is designed to overcome those issues is the MST clustering approach, as presented in Section 6.3. Hence, it will be used during the evaluation process.

Similarly to the previous section, the evaluation is split into two parts. On the one hand, there is a study that is based on the real data set that has been introduced in Section 8.2. Section 10.4.1 is devoted to this particular study. It focuses on the identification of weak collisions in a real network as well as the change of the corrective action plan after eliminating them. On the other hand, we have a simulation study that utilizes the simulation environment that has been introduced in Section 8.1. Section 10.4.2 describes this part of the evaluation, which is primarily focused on studying the impact of removing weak collisions on the network performance.

Each study starts with a motivation as well as a detailed overview of the topics of discussion. Moreover, it lists the parameter selection, e.g., the verification area selection, and concludes with the description of the scenario and the results achieved during the experiments.

10.4.1 Real Data Study

The idea of the real data study is to identify and observe the problem of having weak verification collisions. In particular, the following topics are of high interest:

- ✓ Identification of weak collisions in the real data set.
- ✓ Analysis of the corrective action plan that is generated for the given setup.
- ✓ Analysis of the corrective action plan after eliminating the weak verification collisions.

It should noted that the terms weak verification collision and weak collisions are used as synonyms.

10.4.1.1 Verification Area Selection

The selection of verification areas is made in the same way as described in the verification collision study, i.e., an area is comprised of the cells of the reconfigured adjacency. Section 10.3.1.1 outlines the selection process.

10.4.1.2 CKPIs and Profiling

The profiling procedure resembles the one used during the verification collision study, as described in Section 10.3.1.2. That is, it is based on the z-score of the PM data. However, the vector of CKPIs $\mathbf{k}^{\perp} = (k_1^{\perp}, \ldots, k_n^{\perp})$ is comprised of the following two performance indicators:

- `EUTRAN_RLC_PDU_RETR_R_DL`: retransmission rate for RLC PDUs in downlink direction.
- `EUTRAN_RLC_PDU_RETR_R_UL`: retransmission rate for RLC PDUs in uplink direction.

In contrast to the verification collision study, the retransmission rate for RLC PDUs in downlink direction is added to $\mathbf{k}^{\perp}$. Furthermore, a change is considered as anomalous if the z-score of at least one of those KPIs stays above the threshold of 1.8 for two hours, i.e., two PM granularity periods.

10.4.1.3 Distance Function

The distance function d (cf. Equation 6.6) used to compute the edge weights in the cell behavior graph $G^{\Sigma} = (V^{\Sigma}, E^{\Sigma})$ (cf. Definition 6.2) is given in Equation 10.1. For two anomaly vectors $\mathbf{a}^{\perp} = (a_1^{\perp}, a_2^{\perp}, \ldots, a_n^{\perp}) \in \mathbb{R}^n$ and $\mathbf{a}^{\perp\prime} = (a_1^{\perp\prime}, a_2^{\perp\prime}, \ldots, a_n^{\perp\prime}) \in \mathbb{R}^n$ the Pythagorean formula is applied.

$$d(\mathbf{a}^{\perp}, \mathbf{a}^{\perp\prime}) = d(\mathbf{a}^{\perp\prime}, \mathbf{a}^{\perp}) = \sqrt{\sum_{k=1}^{n} (a_k^{\perp\prime} - a_k^{\perp})^2} \tag{10.1}$$

As a result, the tree T^{Σ} formed by the cell clustering procedure (cf. Algorithm 1, Section 6.3) is an Euclidean MST, i.e., the weight of every edge $w(v_i^{\Sigma}, v_j^{\Sigma})$ equals the Euclidean distance between $v_i^{\Sigma} \in V^{\Sigma}$ and $v_j^{\Sigma} \in V^{\Sigma}$.

10.4.1.4 Edge Removal Function

The edge removal function ξ (cf. Equation 6.7), based on which the forest F^{Σ} in Algorithm 1 (Section 6.3) is formed, is selected as follows:

- The edges of T^{Σ} whose weight exceeds the 99^{th} percentile of all edge weights are removed.

10.4.1.5 Scenario and Results

To begin with, let us recall the results presented in Figure 10.8. It shows the number of adjacency object modification that have been made in the LTE network. In particular, numerous changes are made on the 28^{th} of November, as well as on the 4^{th}, 5^{th}, 10^{th}, and 11^{th} of December 2013.

As a second step, the presence of anomalous objects is studied. Figure 10.11 depicts the number of such objects as well as the number of time slots that are required if an attempt is made to rollback the changes. Compared to the results from Section 10.3.1.3, there is a slight increase in the number of anomalous objects. It is caused by the inclusion of the second KPI in $\mathbf{k}^{\perp}$.

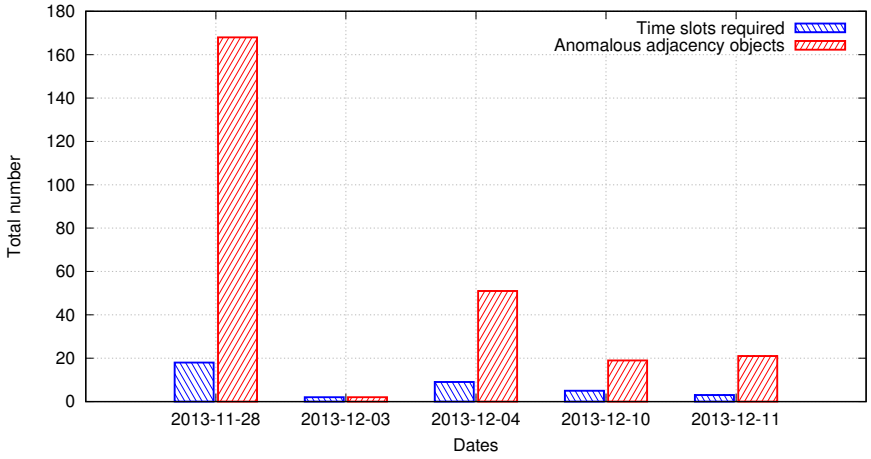
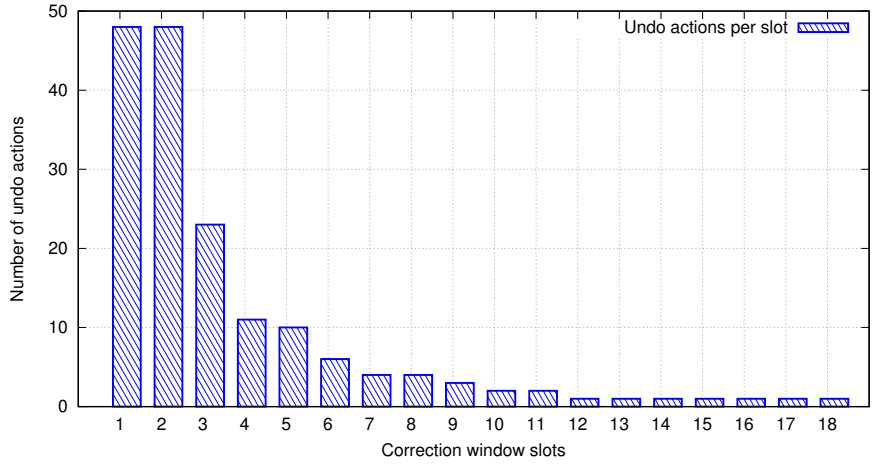
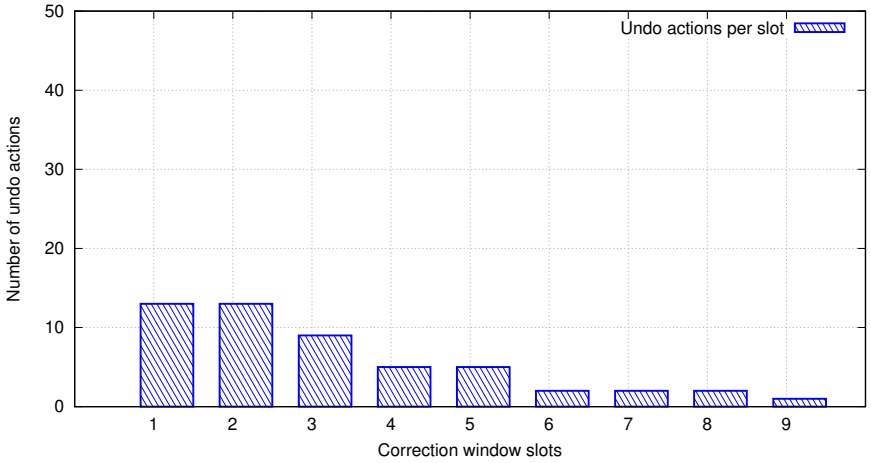


Figure 10.11: Distribution of the anomalous adjacency objects in the real data set

Next, the corrective action plan generated for this given parameter setup is observed. Figure 10.12 visualizes the distribution of the undo actions for the 28^{th} of November and the 4^{th} of December 2013. The x-axis represents the time slots whereas the y-axis the number of undo actions that are allocated to each slot. As shown, the initial processing of the plan for the 28^{th} of November requires 18 time slots, whereas for the 4^{th} of December 9 time slots.



(a) Corrective action plan for the 28^{th} of November



(b) Corrective action plan for the 4^{th} of December

Figure 10.12: Undo action distribution before the elimination of weak verification collisions

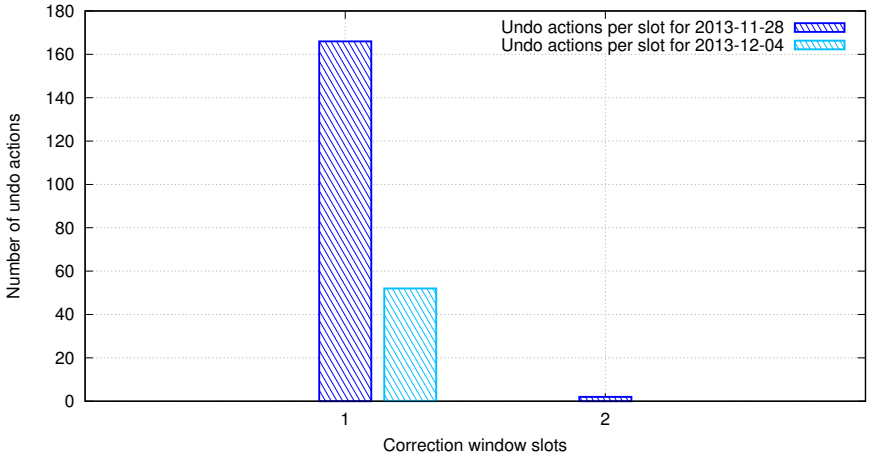


Figure 10.13: Undo action distribution after the elimination of weak verification collisions. The corrective action plan for the 28^{th} of November consist of one block whereas the plan for the 4^{th} of December of two blocks.

Hence, the question that arises is how the distribution of undo actions changes after applying the approach that eliminates weak collisions. Figure 10.13 depicts the resulting corrective action plan. For the 28^{th} of November the verification process requires only two correction window time slots, whereas for the 4^{th} of December just one time slot, which is a significant decrease compared to the initial slot allocation (cf. Figure 10.12).

10.4.2 Simulation Study

The purpose of the simulation study is to observe how the network performance behaves when, on the one hand, weak collisions are eliminated and, on the other hand, such collisions are left over. As we saw in the real data study most undo actions got assigned to the first time slot which is the reason why it is of high interest to monitor the network behavior after deploying those actions. Furthermore, it is of particular interest to observe the network performance when the number of degraded cells varies, and to identify cases where the cell clustering approach does not manage to fulfill its tasks. In summary, the topics discussed in this sections are:

- ✓ The elimination of weak collisions, resolving the valid ones, and studying the cell anomaly level when the number of degraded cells changes.
- ✓ The comparison between CM verification setups that do not utilize the collision elimination and such that use the cell clustering approach.
- ✓ The analysis of the cell anomaly level and the number of deployed undo actions.
- ✓ Observing the interaction with active SON functions.
- ✓ Exploring the limits of the MST-based clustering approach.

10.4.2.1 Active SON Functions

In total, three SON functions are active during the experiments: the MRO, RET, and TXP function. They are required to make reconfigurations while the verification of CM changes is running. As a result, they may interfere with the verification process which may potentially lead to the rollback of changes not harming performance. Moreover, those functions have different optimization goals: the MRO function optimizes the handover of UEs between neighboring cells, whereas RET and TXP adjust the coverage. Thereby, they are inducing changes that may fall within different cluster groups.

It should be noted that the function execution is coordinated, i.e., no two functions adjust the configuration of a cell at the same time.

10.4.2.2 Verification Area Selection

During the simulation study two types of cell changes are deployed. On the one hand, adjustments to the physical cell borders are made. Therefore, to analyze the impact

of such changes, all direct neighbors of the reconfigured (target) cell are added to its verification area.

On the other hand, asymmetrical changes to the CIO are carried out, i.e., a reconfiguration of the offset between the target and the source cell[1], but not vice versa. Also, if a cell is selected for reconfiguration, the handover performance to all neighbors is assessed, i.e., the offset to all neighbors can be changed at the same time. Hence, the verification area is selected in the same way as described above.

10.4.2.3 CKPIs and Profiling

The CKPI vector $\mathbf{k}^{\perp} = (k_1^{\perp}, \ldots, k_n^{\perp})$ consist of three indicators (cf. Section 8.1.1.2):

- Handover Success Rate (HOSR)
- Channel Quality Indicator (CQI)
- Handover ping-pong rate

The first two fall within the success, whereas the last one is a representative of the failure KPI class (cf. Section 5.3.1).

The profiling is done in the same way as introduced in Section 10.1.1.3. That is, the CKPI anomaly level is based on the z-score of the current CKPI values. The training phase is set to 70 simulation rounds.

A verification area is marked as potentially anomalous, i.e., assessed by the verification process, when at least one CKPI anomaly level of at least one cell falls in the range $(-\infty; -2.0]$ for success KPIs and $[2.0; \infty)$ for failure KPIs.

10.4.2.4 Distance Function

Distance function d (cf. Equation 6.6), which is required for the formation of the cell behavior graph G^{Σ} (cf. Definition 6.2), is based on the Euclidean distance. A description is provided in Section 10.4.1.3.

10.4.2.5 Edge Removal Function

The edge removal function (cf. Equation 6.7) is used to form forest F^{Σ} (cf. Algorithm 1 in Section 6.3). Furthermore, there are two implementations of the function:

- Removing all edges from T^{Σ} whose weight exceeds 1.50 the average edge weight.
- Removing all edges from T^{Σ} whose weight exceeds 1.75 the average edge weight.

[1] The terms target cell and source cell come from the handover procedure in LTE. A handover target cell should not be confused with the target cell of a verification area.

10.4.2.6 Compared Strategies

In this experiment, two strategies are compared with each other. On the one hand, there is a configuration that utilizes the collision resolving approach, as described in Sections 6.5 and 6.6. On the other hand, a configuration is defined that makes use of the MST-based clustering approach, as introduced in Section 6.3. It resolves collisions in the very same way as the first one does. The difference, however, is that it eliminates weak collisions before entering the constraint optimization phase.

10.4.2.7 Scenario and Results

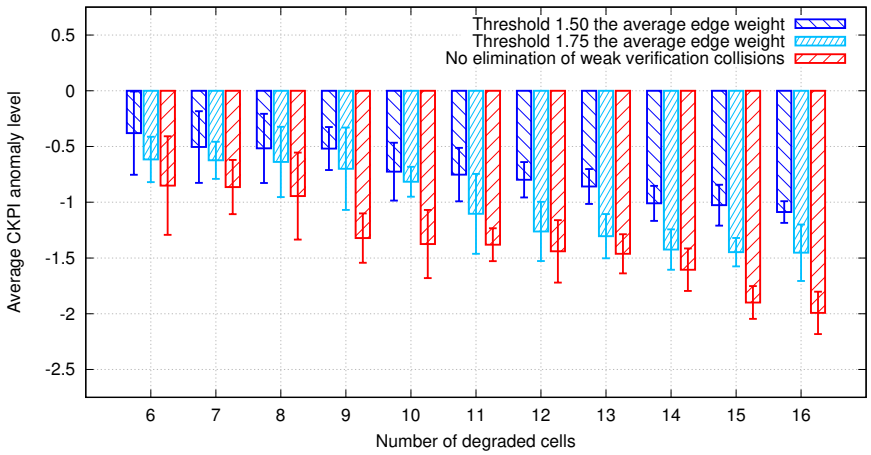
Each experiment is set to last 9 test runs, where each test run is set to last 5 simulation rounds. Before starting a test run, cells are selected for degradation. The selection is made by adding all cell identifiers to a list, permuting the list, where all permutations occur with equal likelihood, and selecting the first n items. The degradation itself is carried out by deploying two untypical configurations. On the one hand, the coverage of half of the selected cells is changed by setting their transmission power to 40 dBm. On the other hand, the handover capabilities of the other half is manipulated by changing their CIO to -5.0.

Furthermore, the total number of experiments is 11. They differ in the number of cells marked for degradation, i.e., the selection of n. The lowest number of degraded cells is 6 whereas the highest is 16. Also, during the experiments all SON functions are allowed to optimize the network by changing CM parameters of interest.

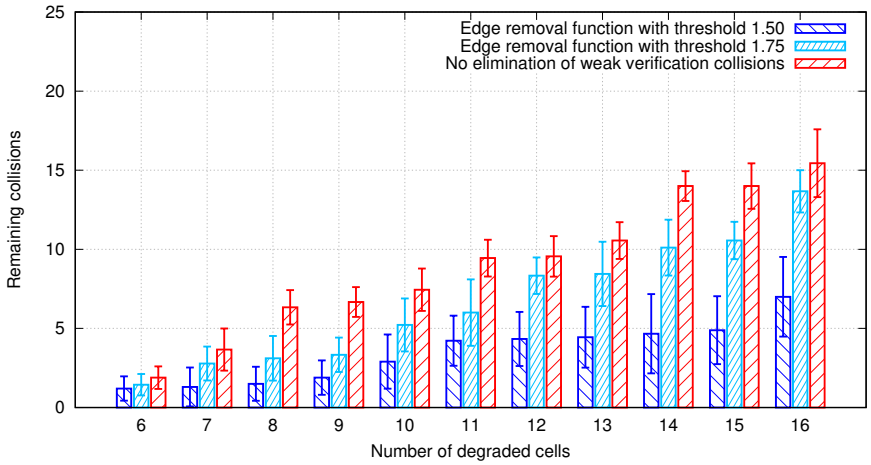
The results are computed in the following way. The negative CKPI anomaly level is taken in the case of failure KPIs, and left as it is for success KPIs. Then, the overall network performance is estimated by averaging all CKPI anomaly levels reported by all cells after processing the first correction window time slot. In addition, the 95% confidence interval around the sample mean is calculated.

The results of this study are shown in Figure 10.14(a). As the observations indicate, the elimination of weak collisions manages to significantly improve the network performance already after deploying the first set of undo actions. Moreover, the performance improves when we lower the threshold to 1.5 times the average edge weight. The reason why this configuration outperforms the remaining ones originates from the number of collisions that got eliminated before starting to process undo actions. Figure 10.14(b) gives more information about this fact. It shows the number of undo actions remaining in collision after processing the first correction window slot. Similarly to the real data study (cf. Section 10.4.1.5), most actions get allocated to the first slot after removing the weak collisions. Hence, there are less entering the verification process afterwards. Remarkably, the configuration that is not utilizing the cell clustering approach gets the worst result, which is mainly due to the high number of collisions that prevent the simultaneous deployment of necessary undo actions.

Finally, some remarks about the limits of the clustering approach should be made. The



(a) Average CKPI anomaly level after processing the first correction window time slot. The higher, the better. A value near zero indicates that the CKPIs are performing as expected.



(b) Number of remaining collisions after processing the first correction window time slot

Figure 10.14: Results of the simulation study of eliminating weak verification collisions

edge removal function ξ has to be used with caution since the deletion of too many edges from the tree T^{Σ} may lead us to the point where we have no verification collisions at all. Consequently, the verification process will start undoing too many changes, including such that did not harm network performance. During the simulation study, this effect occurred when a threshold below 1.5 times the average edge weight was selected.

10.5 Handling Fluctuating PM Data

The idea of this case study is not only to observe whether the network performance has degraded, but mainly to study the problem of having fluctuating PM data. Also, it is of particular interest to monitor the behavior of the CM verification process when it analyzes such PM data. It should be noted that issue itself is discussed in Section 3.2.1.

The study is carried out only by using the simulation environment (cf. Section 8.1). The main reason for not considering the real data set is the fact that the LTE network was not optimized by SON functions that could lead to significant fluctuations in the PM data.

The topics that are discussed in this section can be summarized as follows:

- ✓ The ability of the verification process to dynamically adapt the observation window based on cell PM data.
- ✓ The study of the impact of neglecting PM fluctuations on the ability of the verification process to provide corrective actions.
- ✓ The analysis of the CKPI anomaly level after deploying the suggested corrective actions.
- ✓ The analysis of the impact of PM fluctuations on the SON functions that optimize the network.

10.5.1 Simulation Study Setup

10.5.1.1 Active SON Functions

The SON functions that are active during the experiments are MRO, RET, and TXP, as described in Section 10.4.2.1. The function selection is motivated by the scenario setup. At first, obsolete coverage settings are deployed which are corrected by the SON functions. Their activity may result in PM fluctuations as they try to reach their optimization goal. Then, the CM verification process is triggered to assess changes and rollback those leading to a degradation.

10.5.1.2 Verification Area Selection

The formation of the verification areas is identical to the strategy outlined in Section 10.4.2.2, i.e., it is comprised of the reconfigured cell and its direct neighbors.

10.5.1.3 CKPIs and Profiling

The CKPI vector $\mathbf{k}^{\perp} = (k_1^{\perp}, \ldots, k_n^{\perp})$ is comprised of the following three performance indicators:

- Handover Success Rate (HOSR)
- Channel Quality Indicator (CQI)
- Handover ping-pong rate

The profiling procedure is identical the one presented in Section 10.1.1.3. That is, the CKPI anomaly level is computed as the z-score of the current CKPI value. The total number of simulation rounds required to generate the profiles is 70.

10.5.1.4 Observation Window

At first, the negative CKPI anomaly level is taken for failure KPIs (i.e., the handover ping-pong rate), and left as it is for success KPIs (i.e., HOSR and CQI). Second, the observation $\psi(\mathbf{a}^{\perp})_t$ from Equation 6.5 at time t is computed as follows:

$$\psi(\mathbf{a}^{\perp}) := \frac{1}{n} \sum_{i=1}^{n} a_i^{\perp} \tag{10.2}$$

The state update factor α, required to estimate the cell anomaly level (i.e., the CVSI) $\vartheta(\mathbf{a}^{\perp}, \alpha)$ from Equation 6.4, is selected at time t as follows:

- $\alpha = 0.2$ if $|\psi(\mathbf{a}^{\perp})_t| \in [0; 1)$, i.e., the observation is up to one standard deviation away from the expected performance.
- $\alpha = 0.4$ if $|\psi(\mathbf{a}^{\perp})_t| \in [1; 2)$, i.e., the observation is between one and two standard deviations away from the expected performance.
- $\alpha = 0.8$ if $|\psi(\mathbf{a}^{\perp})_t| \in [2; \infty)$, i.e., the observation is more than two standard deviations away from the expected performance.

Hence, the more unusual the current observation is, the higher the impact on the overall cell anomaly level. The selection of different update factors that depend on the current network state is motivated by the approach used in [LPCS04]. The authors make use of a similar technique to propagate data packets in a sensor network.

Based on those estimations, a verification area $\Sigma^{M'}$ (cf. Definition 5.3) is considered as being anomalous at time t, if and only if the following condition is met:

$$\frac{1}{|\Sigma^{M'}|} \sum_{\sigma \in \Sigma^{M'}} \vartheta_t^{\sigma} \in (\infty; -2.0] \tag{10.3}$$

10.5.1.5 Compared Strategies

For this study, two configurations are defined. On the one hand, we have the CM verification process that has all features enabled (cf. Sections 6.1 to 6.6). That is, verification areas are assessed by utilizing exponential smoothing, weak verification collisions are eliminated by the MST-based clustering approach, valid collisions are resolved by the constraint optimization-based technique.

On the other hand, we have the same setup for which the CVSI feature is disabled. The CVSI feature is turned off by setting the state update factor α to 1. As a result, this configuration should be more aggressive, i.e., it should mark verification areas as anomalous more often.

10.5.2 Scenario and Results

In today's mobile networks it is not uncommon for cells to be supplied with obsolete coverage settings. Typically, it happens when the initial assumptions about the environment significantly change [HSS11]. A significant change can be the result of the construction or demolition of buildings, the insertion of new base stations, and seasonal changes. As a result, the coverage can be reduced compared to what it is possible to achieve with optimal settings.

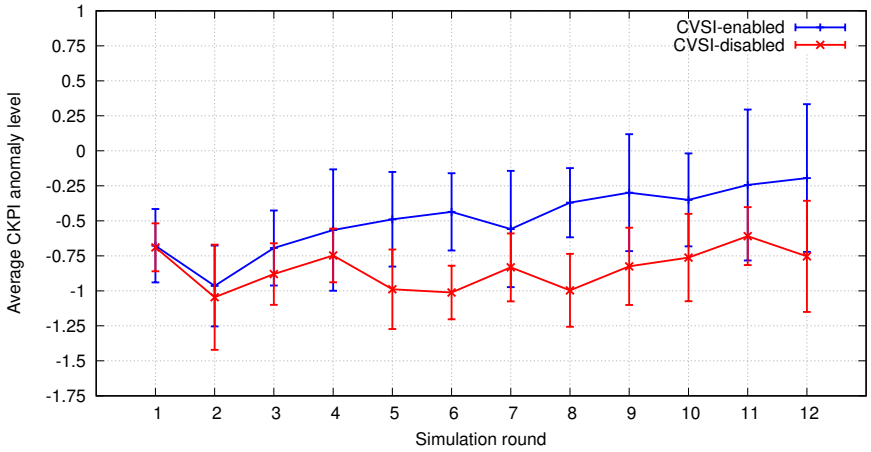
To replicate such a situation, obsolete configurations are applied to nine cells. The antenna tilt of four cells is set to 0.0 degrees, the tilt of another two cells is set to 1.0, the degree of another two cells is set to -4.0, and the tilt of the remaining cell is 3.0. The transmission power is not changed, i.e., it is set to the maximum of 46 dBm. Those settings are applied before starting a test run. In total, seven test runs are carried out. The duration of a single test run is 12 simulation rounds. During a test run the SON functions are allowed to optimize the network.

Furthermore, the function activity is managed by the SON coordinator. The coverage optimization functions have a higher priority than the MRO function. Hence, MRO is suppressed if RET or TXP wish to make a change. All SON functions, however, can be interrupted by the verification process if it decides to execute an undo action, i.e., the verification process is assigned with the highest priority.

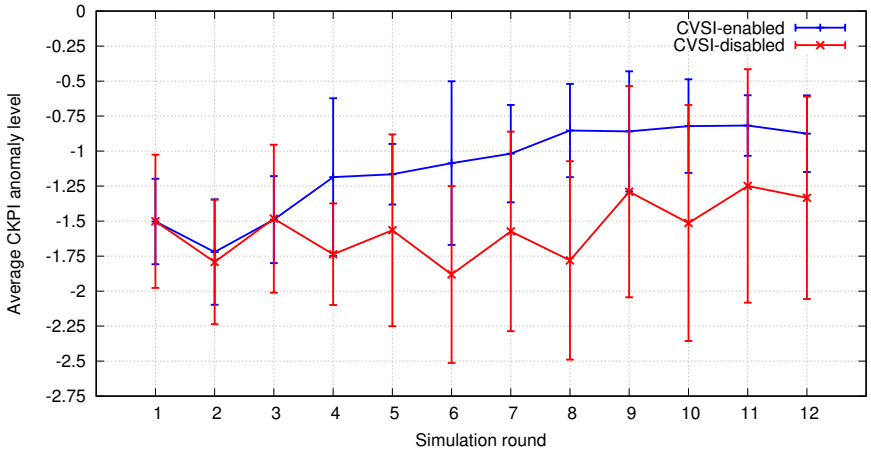
The first observations of this study show that the optimization functions put their highest focus on two cells: 2 and 6. The verification areas formed around those cells include five and six cells, respectively. As shown in Figure 8.4, the two areas are located very close to each other and, therefore, share cells. Hence, it creates the potential for verification collisions.

Figures 10.15(a) and 10.15(b) show the average CKPI anomaly level of the two areas. The anomaly level changes frequently which was caused by fluctuating PM data. As expected, the SON functions were immediately activated and tried out different CM settings in order to reach their optimization goal. Concretely, the RET function started to adjust the antenna tilt, which was in most cases followed by a CIO change. The CIO change was triggered by MRO. Those changes also induced temporal performance drops which forced the verification process, that utilizes the second configuration (disabled CVSI feature), to interrupt the ongoing optimization. As a result, the functions were required to reapply their changes which caused the average CKPI anomaly level of both areas to fluctuate and never reach zero.

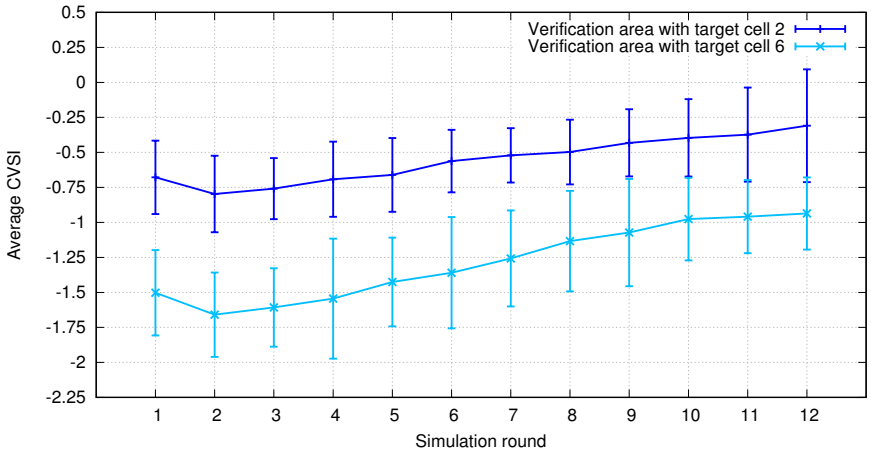
On the contrary, the configuration that utilizes the CVSI feature manages to provide better results. At the end of simulation round 12, the CKPI anomaly levels return to the expected range, i.e., a value near zero. Figure 10.15(c) visualizes why we are able to get such results. After simulation round 4, the CVSI of both areas does not fall below the threshold. Hence, they are not further assessed by the verification process and are let to be optimized by the SON functions.



(a) Average CKPI anomaly level of the verification area of target cell 2. The higher, the better. A value near zero indicates that the KPIs are performing as expected.



(b) Average CKPI anomaly level of the verification area of target cell 6. The higher, the better. A value near zero indicates that the KPIs are performing as expected.



(c) Average CVSI of the verification areas. The higher, the better.

Figure 10.15: Results of the simulation study of handling fluctuating PM data

10.6 Topology Verification

The last part of the evaluation chapter is dedicated to the verification of concurrent CM and topology changes. Generally speaking, turning cells on or off may induce anomalies which may lead to incorrect corrective actions while verifying configuration changes. Furthermore, adding or removing a cell may generally lead to incomplete profiles (cf. Definition 3.12) that can further cause significant anomalies in the behavior of already enabled cells. Note that Section 3.2.5 discusses those problems in detail.

For this reason, the verification process is split into two parts, as introduced in Section 5.1. On the one hand, we have the CM verification process which was up to now the major topic of discussion in this chapter. On the other hand, there is the topology change verification process which is described in detail in Chapter 7. It is based on Steiner trees, i.e., MSTs whose costs can be reduced by adding extra vertexes to the initial input graph. Those nodes are referred to as Steiner points which represent on-demand cells.

The Steiner tree-based verification approach also makes use of helper functions that influence the algorithm's outcome. In particular, we have the edge weighting function that is required to form the topology verification graph (cf. Definition 7.3), and the Steiner point assessment function (cf. Equation 7.3) which converts an on-demand cell to a Steiner terminal, i.e., a vertex that is always used for the formation of the Steiner tree. Furthermore, the approach permits the penalization of cells that are frequently being turned on and off. Hence, the topics discussed in this section can be summarized as follows:

- ✓ Study and evaluate the distance function used to form the topology verification graph.
- ✓ Observe the impact of the Steiner point assessment function on the outcome of the topology verification process.
- ✓ Evaluate the cell penalization capabilities of the topology verification process.
- ✓ Compare the Steiner tree-based verification approach with another state-of-the-art verification strategy.
- ✓ Study the limits of the Steiner tree-based verification approach.

The study itself is carried out by using the simulation environment, as described in Section 8.1. Further, it is split into two parts. Section 10.6.1 lists the setup, i.e., the active SON functions, the verification area selection, the KPI selection, the profiling, as well as the functions used by the topology verification algorithm. Section 10.6.2 presents the results of the study.

10.6.1 Simulation Study Setup

10.6.1.1 Active SON Functions

The SON function that is active during the experiments is RET, as introduced in Section 8.1.2. Furthermore, a basic cell ESM feature is activated. It turns on a cell when its load is above a threshold and off otherwise.

10.6.1.2 Verification Area Selection

There are two types of verification areas that are generated during the experiments. On the one hand, there are areas which are formed around the cell that has been reconfigured, i.e., the antenna tilt degree has been adjusted. Such areas are comprised of the reconfigured small cell and its first degree neighbors. The motivation for selecting them in such a way is outlined in Section 10.3.2.2.

On the other hand, we have areas that are defined by the topology verification process. Verification areas of this type are formed in two steps. First, around every small cell a sub-area is formed that includes the cell itself and its first degree neighbors. Second, all sub-areas that share common cells are united into one single verification area. As a result, we get a connected graph as required by Definition 5.3.

10.6.1.3 CKPIs and Profiling

The CKPI vector $\mathbf{k}^{\perp} = (k_1^{\perp}, \ldots, k_n^{\perp})$ used by the CM verification process includes the following performance indicators:

- Channel Quality Indicator (CQI)
- Handover Success Rate (HOSR)
- Handover ping-pong rate

The profiling procedure and the cell anomaly level selection is identical to the one introduced in Section 10.4.2.3.

10.6.1.4 TKPIs and Profiling

The TKPI vector $\mathbf{k}^{\vdash} = (k_1^{\vdash}, \ldots, k_n^{\vdash})$ required by the topology verification process is comprised of the following performance indicator:

- Cell load based on the PRB utilization

The profile generation and anomaly level computation are identical to the procedure described in Section 10.1.1.3. An element $\dot{p}_i^{\vdash}$, where $i \in [1, n]$, of a TKPI profile vector $\dot{\mathbf{p}}^{\vdash} = (\dot{p}_1^{\vdash}, \ldots, \dot{p}_n^{\vdash})$ is a sequence of t observations which are collected while the network is operating as expected. The outcome of function $\dot{\varphi}$ (cf. Equation 7.1) is calculated by

taking the sequence of observations of $\dot{p}_i^{\vdash}$, as well as the current $\dot{k}_i^{\vdash}$, and computing the z-score of all values. The z-score of the current TKPI gives us its anomaly level.

It should be noted, though, that only static cells have a profile. The reasons have been discussed in detail in Section 3.2.5. As a result, function $\mathring{\phi}$ (cf. Equation 7.2) computes the TKPI anomaly levels of on-demand cells by taking the weighted sum of the anomaly levels of its direct static neighbors. That is, the outcome is the weighted sum of the load anomaly levels. The weight itself is computed by taking the served UE ratio, i.e., the number of UEs served by a direct neighbor divided by the total number of UEs served within the verification area.

10.6.1.5 Edge Weighting Function

In order to prevent edge weights $\in (-\infty, 0]$, all load anomaly level values are put within the interval of $[1; 2]$. The weight of an edge in the topology verification graph $G^{\mathcal{T}}$ (cf. Definition 7.3) is computed by summing up the load anomaly level of the cells represented by adjacent vertexes. In addition, if one of the cells is an on-demand cell, only a portion of the edge weight is taken by multiplying it with a *Steiner edge factor*.

10.6.1.6 Steiner Point Assessment Function

The Steiner point assessment function ι (cf. Equation 7.3) is based on exponential smoothing of the TKPIs, in particular, the cell load. In case the smoothed cell load value falls below a Steiner threshold of 15%, an on-demand cell is considered as a Steiner point, otherwise as a terminal. The update factor ranges between 0.1 and 1.0.

10.6.1.7 Steiner Point Penalization

If an on-demand cell gets enabled and shortly after that disabled, the Steiner edge factors of all edges, leading to the vertex repressing it, are increased by using a step size of 0.1. Over time, this factor gets decreased by the same step size until it reaches the initial value. In addition, on-demand cells that have become terminals are rewarded by immediately setting the Steiner edge factor to the initial value.

10.6.1.8 Compared Strategies

In total, two configurations are compared against each other. At first, we have only CM verification, as presented in Chapter 6. This setup represents the default verification strategy. It monitors the activity of all SON functions, including features optimizing the energy consumption, and rolls back the changes harming performance.

Second, we have the Steiner tree-based verification, as introduced in Chapter 7. For this setup, cells that are verified by the Steiner tree-based verification algorithm are not processed by the CM verification process.

10.6.2 Scenario and Results

The experiments are carried out by using the cell deployment that has been introduced in Figure 8.5 (cf. Section 8.1.1.5). In total, there are 32 LTE macro and 9 small cells which are assessed by the verification process after the completion of a simulation round.

10.6.2.1 Edge Weighting Function

The evaluation starts by estimating how function $d^{\mathcal{T}}$ impacts the Steiner tree $T^{\mathcal{T}}$. In particular, we are interested in how the factor used to multiply the weight between an on-demand and another cell affects the outcome. In order to induce an unusually high load at the macro cells, a user group that consists of 150 UEs is added to one particular part in the network. The area itself is covered by on-demand cells 33, 34, and 35.

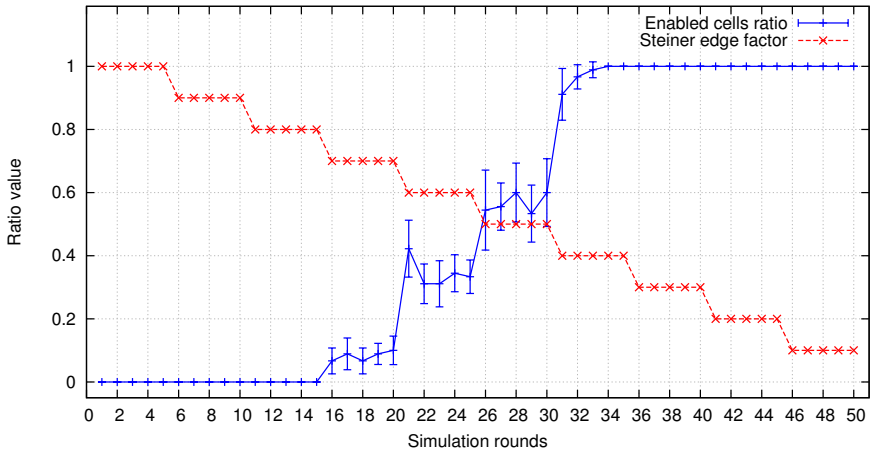
The experiment itself starts by selecting a Steiner edge factor of 1.0, which is decreased by 0.1 after every fifth simulation round. Note that all cell states are reset as soon as the factor gets decreased. Figure 10.16(a) shows the results, in particular, the percentage of on-demand cells that get enabled for the given Steiner edge factor. As shown, for a factor between 1.0 and 0.8, none of the on-demand cells are switched on, i.e., no Steiner points are added to the graph. After decreasing its value to 0.7, the first on-demand cells are allowed to be switched on. When selecting a factor of 0.6, approximately 33% of all on-demand are enabled. A factor of 0.5 allows roughly 60% of those cells to be added to the Steiner tree, whereas a factor of 0.4 and lower permits all on-demand cells to become active.

The question that arises here is how the load anomaly level changes when selecting a different Steiner edge factor. Figure 10.16(b) shows the average load anomaly level of all on-demand cells and all of their static neighbors. As presented, for a Steiner edge factor of 0.6 or less, the z-score-based anomaly level ranges within approximately the same interval. Note that all figures represent 95% confidence intervals that are computed around the sample mean of a certain number of consecutive test runs. Here, the total number of test runs is 10.

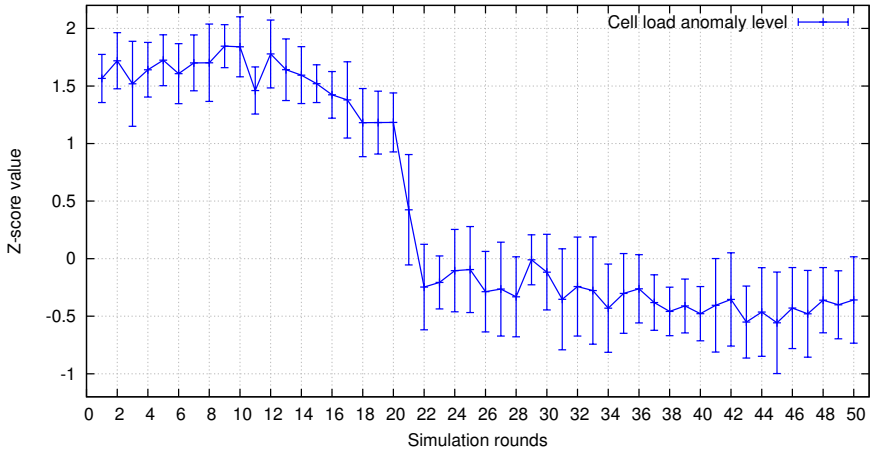
10.6.2.2 Steiner Point Assessment Function

Next, we are going to test function ι, which basically converts a Steiner point to a terminal and vice versa. Since it is based on exponential smoothing, we will evaluate the parameter used to update the current value. The experiment starts with an update factor of 1.0 which is gradually decreased by 0.1. In contrast to the previous setup, the evaluation window is set to 10 rounds. The setup is identical as before, however, after the fifth round the UE group is removed from the network. Afterwards, the simulation rounds are counted during which no longer necessary on-demand cells stay switched on.

Figure 10.17 gives us the results of this observation after 10 test runs. For a factor of 1.0, the on-demand cells are almost immediately disabled after removing the UE group. It is indicated by the low percentage of simulation rounds during which cells remain



(a) Correlation between the Steiner edge factor and number of enabled cells



(b) Impact on the cell load anomaly level

Figure 10.16: Evaluation of the edge weighting function

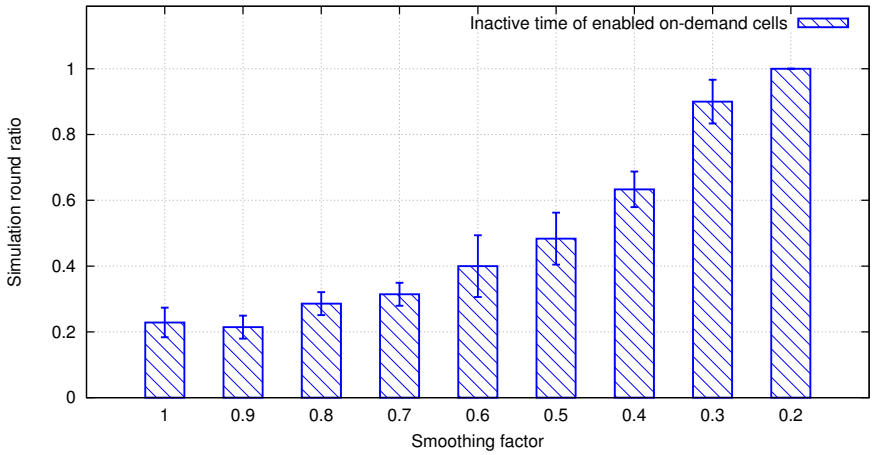


Figure 10.17: Evaluation of the Steiner point assessment function

turned on. However, as we start decreasing the update factor, more on-demand cells stay switched on. This trend can be seen for any factor in the interval between 0.3 and 0.9. For a factor of 0.2 none of the cells are switched off, even though the UE group has disappeared.

10.6.2.3 Steiner Tree-Based Verification

To compare the two configurations, the RET function is allowed to optimize the antenna tilt of 16 of the 32 macro (static) cells. Those cells have at least one small (on-demand) cell as neighbor. In addition, in order to trigger the wake up or sleep mechanism of the small cells, four UE groups are added. As described in Section 8.1.1.4, the groups consist of 150, 75, 85, and 120 users. Thereby, simultaneous CM and topology changes should emerge.

In total, 7 test runs, each lasting 20 simulation rounds, are carried out. Moreover, after every fifth round a new UE group is randomly selected. The selection is made by adding all UE groups to a list, permuting the list, where all permutations occur with equal likelihood, and selecting the first item. Every time a new UE group is selected, the average load anomaly level of each of the 16 macro cells is measured. Hence, we have 28 samples for each of those cells. It should be noted that the Steiner edge update factor is set to 0.6 whereas the smoothing update factor to 0.5. The selection is motivated by the observations made in Sections 10.6.2.1 and 10.6.2.2.

Figure 10.18 visualizes the results. As outlined, the Steiner tree-based verification approach manages to improve the anomaly level of all 16 cells, i.e., putting it near zero which is the expected state. Especially the anomaly level of cells 6, 9, 10, 18, and 24 is significantly changed. On the contrary, using only CM verification leads to a worse anomaly level, which is caused also by the undo of RET changes. Those changes were blamed although they did not do any harm.

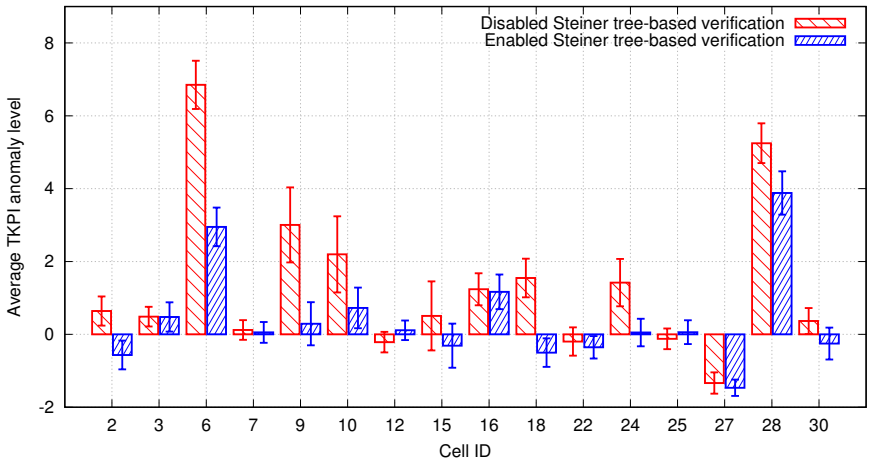


Figure 10.18: Evaluation of the Steiner tree-based verification approach

Finally, a statement about the limits of the Steiner tree-based verification approach should be made. First of all, the TKPI vector has to be set properly. The approach is able to solve topology-related issues only if KPIs that reflect the utilization state of a cell are taken into consideration. Second, the outcome also depends on the network topology. For instance, in Figure 10.18 some cells are more than two standard deviations away from the expected mean even though the topology verification process was triggered. Those macro cells are overloaded due to the limited number of neighboring small cells which simply could not take over more traffic. Hence, care should be taken while planning the network and deploying on-demand cells. Third, the introduced approach depends on its helper functions, which was also the reason why they got much attention in this section. Setting them up inappropriately may result in suboptimal corrective actions.

10.7 Summary

This chapter provides an extensive evaluation of the verification concept that is presented in Chapters 5-7 and whose implementation is discussed in Chapter 9. The evaluation itself is split in two parts: one that evaluates the concept's CM verification capabilities and another that is dedicated to the study of the ability to verify dynamic topology changes.

In summary, the first part of the evaluation has shown that in most cases SON functions are not able to verify their own actions. It was also shown that neglecting verification collisions leads to performance degradation and that resolving them considerably improves network performance. Furthermore, it is also shown that the elimination of weak collisions substantially improves the corrective action decision process. The importance of handling fluctuating PM data is highlighted as well.

A detailed summary of those results is provided by the answers to the questions (contributing to **O3.2** and **O3.3**) listed below.

O3.2: Study and evaluate the impact of uncertainties on the outcome of a verification process.
Q: What is the impact of neglecting verification collisions?
A: In Section 10.2, a detailed analysis is made of the impact of neglecting verification collisions on the network performance. That is, rolling back changes without resolving the uncertainties that emerge during the process of verification, as given by Definition 3.3. The evaluation itself is a comparison between two configurations, both utilizing the CM verification process (cf. Chapter 6) and a SON coordinator. They differentiate in the way of how they handle verification collisions: the first relies only on the SON coordinator to resolve known (pre-defined) conflicts, as presented in Section 2.3.2.3, whereas the second one makes use of the verification collision resolving approach, as introduced in this thesis. The results show that for a high number of degraded cells (cf. Section 10.2.2) the CM verification process manages to improve the cell anomaly level by up to 50%.

Q: Does the presence of fluctuating PM data result into verification collisions?
A: Yes. Fluctuating PM data may result into unnecessary undo actions and, therefore, into verification collisions. As outlined in Section 10.5, those unnecessary actions may also interrupt a SON optimization process and prevent cells from reaching their performance optimum. Although a SON function may induce a temporal performance decrease, its changes must not be immediately undone.

Q: Can verification collisions emerge in a mobile networks not fully implementing SON features?
A: Yes. Verification collisions may emerge when verifying any types of configuration changes, including such that are manually made or computed by offline algorithms. The observations on the real data set made in Sections 10.3.1 and 10.4.1 confirm this fact. Although a relatively small verification area was selected, numerous collisions emerged during the process of verification. The results are presented in Sections 10.3.1.3 and 10.4.1.5.

Q: How do weak verification collisions reflect on the verification process' outcome?
A: Section 10.4.1 is dedicated to the elimination of weak collisions in the real data set whereas Section 10.4.2 in the study of the effects after their removal. The latter one was carried out in the simulation environment. Both studies confirm that the presence of weak collisions increases the size of the corrective action plan (cf. Definition 3.4). As a result, the likelihood of getting an over-constrained plan (cf. Definition 3.7) increases, i.e., it is impossible to process all undo actions in time. Thereby, the cell anomaly level never reaches zero, that is, the performance of the network never falls within the expected ranges.

Q: Are the verification capabilities of SON functions limited?
A: Yes. The observations made in Section 10.1 confirm this statement. SON functions do not have a wide view on the network and are only interested in reaching their own objective. A sub-optimal decision made by one function results in sub-optimal decisions taken by the other active SON functions. Furthermore, if such actions result in a degradation, the network is either not able to completely return to the expected performance state, or requires a lot of time to do so. In particular, the required time was approximately 20 hours for the given setup (cf. Section 10.1.1).

O3.3: Resolve and eliminate uncertainties and provide accurate corrective actions when verifying configuration changes.
Q: How much does the performance improve when weak collisions get eliminated?
A: As the results from the real data study in Section 10.4.1.5 show, the elimination

of weak collisions considerably changes the corrective action plan. Most of the undo actions get allocated to the first correction window slots. Furthermore, the number of plan blocks decreases significantly. The simulation study (cf. Section 10.4.2.7) shows that the removal of such collisions improves the cell anomaly level by up to 50%.

Q: Can the MST-based clustering approach that is used for the elimination of weak verification collisions fail?
A: Yes. In particular, the approach highly depends on the edge removal function as defined in Equation 6.7. As the results in Section 10.4.2.7 show, the removal of edges in the verification collision graph improves the network performance only up to a certain point. Removing an edge means that a collision is no longer considered as valid, i.e., the likelihood of rolling back changes not harming performance increases.

Q: Does the verification concept benefit from the usage of soft constraints?
A: Yes. The usage of soft constraints to identify soft verification collisions (cf. Definition 3.8) allows us to find a solution to an over-constrained verification collision problem. Concretely, the real data evaluation (cf. Section 10.3.1.3) demonstrates the ability of the verification concept to allocate all undo actions to the available correction window slots. The simulation-based evaluation (cf. Section 10.3.2.7) shows that the cell anomaly level significantly improves when we make use of soft collisions that allow the generation of a collision-free (cf. Definition 3.5) and gain-aware (cf. Definition 3.6) corrective action plan.

Q: How does fluctuating PM data impact the corrective action decision process?
A: The results from Section 10.5.2 show that SON functions may induce temporal performance drops which lead to fluctuating PM data. As a result, verification areas are assumed to have degraded and are unnecessarily being processed by the verification mechanism. Furthermore, if the latter one generates undo actions for those areas, ongoing SON optimization processes get interrupted and the affected network areas never reach their performance optimum.

Q: How much do SON functions benefit from the introduced verification concept?
A: There is a twofold answer to this question. First, the experiments made in Section 10.5 show that the SON verification process prevents the interruption of a SON optimization process. As a result, functions reach their performance goal.
Second, the results from Section 10.1 clearly show the limited ability of functions to verify their actions in a SON environment on their own. Having an external entity, that performs this operation, which also has a wider view on the network, leads to an anomaly level near zero.

The second part of the evaluation is dedicated to the topology verification process. In particular, the focus is put on the Steiner point assessment and cell penalization capabilities. Furthermore, the ability to generate corrective topology actions is evaluated. In summary, the results have shown that it is of high importance to handle topology changes in a different way compared to other CM changes. They also show that in the case of dynamic topology changes undo actions are insufficient to provide an acceptable network performance level. The answers listed below (contributing to **O4.2** and **O4.3**) give a more detailed overview of the results.

O4.2: Study and evaluate the impact of topology changes on the process of verifying configuration changes, as well as identify the necessary conceptual changes of a verification process.
Q: Do dynamic topology changes impact a verification process?
A: Yes. As the experiments from Section 10.6 show, dynamic topology changes lead to anomalies in the TKPIs. Thereby, CM changes that are occurring within the same area are rolled back, even though they did not harm performance. In other words, we get a weak corrective action plan (cf. Definition 3.13).

O4.3: Enable topology verification, resolve uncertainties, and provide corrective actions.
Q: What is the benefit of the introduced Steiner tree-based verification approach?
A: The Steiner tree-based verification approach implements the missing component of a strategy that verifies ongoing network changes. In particular, it allows the verification of dynamic topology changes and the generation of a topology corrective action if required. The results from Section 10.6.2 clearly show that the Steiner tree-based approach improves the overall cell anomaly level.

Q: What care should be taken in order to enable topology verification?
A: The Steiner tree-based verification approach cannot meet the topology verification requirements without understanding the effects of the edge weighting function (cf. Section 10.6.2.1), as well as the Steiner point assessment function (cf. Section 10.6.2.2).

Q: What are the limits of the Steiner tree-based verification approach?
A: The main limitation comes from the network topology. As shown in Section 10.6.2.3, the Steiner tree-based verification approach is able to significantly improve the overall anomaly level. Nevertheless, if the network is not able to handle all of the generated traffic, e.g., due to a low number of small cells, we will never get an anomaly level near zero.

Part V

Conclusion

Chapter 11

Conclusion and Future Directions

This chapter concludes as well as summarizes the key findings of this thesis. Concretely, Sections 11.1 and 11.2 highlight the research problems and the developed verification concept, i.e., the CM and the topology verification process. Section 11.3 contributes to the summary by outlining the research results whereas Section 11.4 by discussing future research directions and the future development of the SON verification concept. In the latter case, a new communication handling mechanism is proposed and extensive study of the utilized algorithms in future deployments is suggested.

Published work Future research directions of the concept of SON verification have been published. In particular, the proposed communication handling mechanism can be found in [TAT16].

11.1 Research Problems

Today, mobile communication networks have become complex systems that require automation mechanisms for configuration, optimization, and troubleshooting. Implementing those systems as SONs is one possible way of achieving a high level of automation. Nonetheless, automation does not immediately guarantee a flawless network operation. There are many reasons why automated reconfiguration processes may change CM parameters that are suboptimal or even harm performance. For example, online SON algorithms have a limited view on the network. Some of them may optimize the network only locally, e.g., only the handover parameters between two cells. Offline SON algorithms are usually comprised of sophisticated algorithms, but utilize simulation tools which may produce suboptimal results if the used model is inaccurate.

For this reason, troubleshooting as well as anomaly detection and diagnosis approaches have been developed and used in the area of mobile network management. Unfortunately, they are not always able to provide appropriate corrective actions, as it is extensively discussed throughout this thesis. First and foremost, the ability to generate the most appropriate corrective action depends on whether the used approach is supplied with

accurate diagnosis knowledge. Finding it is a challenging task because networks are manifold, i.e., they generate different types of PM data, and even react in a different way to the same corrective action. In most cases it is even impossible to reuse the results from one observation for the diagnosis on a different RAT, even within the same network.

Second, dynamic topology changes as well as potentially resulting incomplete profiles may complicate the process of finding corrective actions. As stated in Section 3.2.5, a profile specifies how a cell should usually behave, e.g., during peak hours. However, frequently switching cells on or off may invalidate the initially made assumptions about the network.

Nonetheless, even if we neglect the issue of dynamic topology changes, we still may face so-called verification collisions. As given by Definition 3.3, a collision is an uncertainty which configuration change to rollback. It may result in the serialization of the corrective action deployment process, i.e., we have to deploy corrective actions one after the other. Hence, the presence of numerous collisions may further lead to a delay while restoring configurations and even to the inability to completely restore network performance.

Besides those issues, there is also the problem of having an over-constrained corrective action plan. It emerges when the time to process the corrective action plan is not sufficient (cf. Definition 3.7). Furthermore, there are also weak collisions (cf. Definition 3.9) which could additionally impair the process of restoring a cell's configuration .

11.2 The Concept of SON Verification

In this thesis, the concept of SON verification has been presented. It aims to verify configuration changes that have been made in the network by assessing their impact on network performance. Those that harm performance are marked for further analysis and are eventually rolled back. This process has been also introduced as CM verification. Furthermore, the concept of SON verification is also able to verify dynamic topology changes, e.g., those emerging when the network is optimized by energy saving algorithms. The corrective action here is to enable or disable cells depending on their ability to improve the stability of the network.

11.2.1 CM Verification

CM verification is split into three phases. First, the scope of verification is fragmented by splitting the network into so-called verification areas. They represent cells that are of particular interest due to CM changes. Elements from graph and set theory have been utilized to describe this process.

During the second phase, the performance of the verification areas is assessed. To achieve that, a cell behavior model is proposed that represents each cell in the $\mathbb{R}^n$ space by taking its CKPIs into account (cf. Definition 5.5). Furthermore, a strategy that is based on exponential smoothing is applied to limit the impact of fluctuating PM data on

the outcome of the verification process. Then, each area that is marked as degraded is processed by the developed MST clustering technique. It has the purpose of eliminating weak verification collisions by changing the initially defined verification areas. The problem of resolving the remaining collisions is modeled as a constraint optimization problem that makes use of so-called hard constraints. The outcome is a corrective action plan, in particular, a partition of the set of all undo corrective actions. Those having the highest chance of restoring the network performance are allocated to the first block of the partition. Moreover, the optimization also makes use of so-called soft constraints in case the given verification problem is over-constrained, e.g., due to time limitations. They allow to find a solution, i.e., a corrective action plan, that minimizes the total constraint violation and, at the same time, guarantees the collision-free and gain-aware property of the plan (cf. Definitions 3.5 and 3.6).

11.2.2 Topology Verification

Similarly to CM verification, the process responsible for verifying topology changes is also divided into three phases. During the first phase, verification areas are generated. In contrast to CM verification, those areas are formed around cells where topology changes take place. For example, areas comprised of small cells are of particular interest since they are only activated when numerous users enter the network. In order to detect such areas, the behavior of the cells is modeled by using their TKPIs (cf. Definition 5.6). Similarly to CM verification, the behavior is defined in the $\mathbb{R}^n$ space. However, in contrast to CM verification, the behavior of cells not having a profile is defined by the behavior of their direct neighbors.

The problem itself of finding the appropriate corrective actions is represented as a Steiner tree problem, i.e., finding an MST by potentially including an extra set of nodes, also known as Steiner points. Those Steiner points represent on-demand cells (cells that can be disabled) whereas the initial vertexes of the graph represent static cells (cells that are always switched on). An on-demand cell is turned on only if the corresponding Steiner point is selected to form the tree. The presented approach is also accompanied by functionalities that penalize cells that are switched on or off due to fluctuating PM data.

11.3 Results

The evaluation of the developed verification concept studies several aspects. First of all, the ability to verify CM changes and assemble a corrective action plan is studied. The evaluation itself is devoted to the study of the limits of SON functions to verify their own activity, and the observation of the impact of neglecting verification collisions. It is shown that SON functions have a limited ability to verify their operation and are, therefore, not always able to find an appropriate corrective action. The usage of CM verification, however, manages to provide a suitable set of actions and stabilize the performance of the network.

The evaluation also considers the ability of the concept to resolve verification collisions, detect such being weak, as well as handle an over-constrained verification collision problem. Those observations are made by using not only the simulation environment, but also a data set that was generated by a real LTE network. The results show that verification collisions can occur in a mobile network and that the provided solution is able to generate a corrective action plan that restores cells to a previous stable state.

In addition, during the evaluation a closer look is taken at the capabilities of handling fluctuating PM data. The results show that the presence of such data can lead to unnecessary processing of verification areas and, in the worst case, the interruption of a SON optimization process.

In this thesis, the topology verification process is evaluated as well. It is shown that it manages to provide the necessary corrective actions, even when CM changes are made at the same time by other SON functions.

11.4 Future Directions and Work

The evolution of the presented verification concept depends on the direction mobile communication systems are going to take. In particular, it becomes of high interest to follow the development of systems utilizing self-organizing paradigms. Hence, let us observe the trends of such systems before going into details about the future directions of the presented concept.

11.4.1 Evolution of Mobile Networks

In future standards, like the fifth generation of mobile communications, also abbreviated as 5G, advanced techniques will not only apply to physical NEs, but will enable operators to balance load in a multi RAT environment, and further develop traffic steering and dynamic spectrum allocation [Eri14]. Furthermore, a wider variety of use cases are going to emerge [Nex15]. For example, future mobile networks will provide broadband access in dense areas, allow high user mobility, as well as be able to handle extreme real-time communications. Also some base stations will start having a range similarly to commonly used Wi-Fi routers [CZ15].

Network management automation is also going to experience changes induced by 5G technologies [MDM+16], i.e., the SON concept as it is today will be much further developed. First and foremost, a mobile network has to provide interoperability with legacy SON, in particular, with legacy SON functions which typically reconfigure NEs based on static rules. Due to this property, legacy functions may induce conflicting configuration changes as well as increase the likelihood of anomalous cell behavior. In contrast to 4G systems, SON coordination and management will only be able to partially provide a solution due to the complexity and the flexibility requirements of future standards.

Second, there will be a redefinition of SON use cases. For example, current handover optimization functions (like MRO) will not be applicable since future mobile networks will be highly heterogeneous and dense. Furthermore, most of the traffic load is going to be carried by small cells which leads to frequent handover procedures and complex configurations.

Third, future SON functions, also referred to as cognitive functions, will incorporate sophisticated machine learning algorithms and will cover a much wider variety of use cases. Such functions, however, would also need to operate with legacy SON functions since an immediate full transition is unlikely to happen.

11.4.2 Challenges for Verification Approaches

In Chapter 4, approaches that are related to the concept of SON verification has been presented. In total, three categories have been identified: pre-action analysis, post-action analysis, and post-action decision making. Within the first category fall approaches that focus on the avoidance of potential conflicts which may induce undesired network behavior. The introduced methods can be triggered at design- or at run-time. Design-time approaches, like harmonization and co-design (cf. Sections 4.1.1.1 and 4.1.1.2), will continue experiencing the same problems and will even be further limited in their conflict prediction capabilities. The main cause is going to be the wide variety of configuration parameters and increased complexity of the network. Run-time methods, e.g., such being responsible for dynamic action execution and configuration management (cf. Sections 4.1.2.1 and 4.1.2.2), are going to experience scalability issues in future setups. The same applies for adaptive resource allocation (cf. Section 4.1.2.3). In addition, the complexity and the flexibility requirements may have a negative impact on them as well.

Representatives of the second category, post-action analysis (cf. Section 4.2), will mainly experience difficulties in determining corrective actions when numerous automatic reconfiguration entities are active, e.g., the aforementioned SON/cognitive functions. As discussed in Section 11.4.1, the number of such functions will increase and new SON use cases will emerge. As a result, the capabilities of predicting the combined performance impact of each CM change combination, as introduced in Section 3.1.1, will become even more challenging.

Approaches falling within the third category, post-action decision making (cf. Section 4.3), will face the same issues as those experienced by post-action analysis methods. However, since they also actively perform changes, they will need to provide interoperability with legacy SON which, as mentioned in Section 11.4.1, reconfigure the network based on static rules.

11.4.3 Future Work and Open Issues

As we saw in the previous sections, there are numerous factors that are going to impact strategies that follow verification principles. In order to meet the new requirements, the

concept of SON verification would require research in three areas. First, the communication with the active SON/cognitive functions, which is closely connected with the scope of verification. Second, the study of the utilized algorithms will be of high importance. Third, the application of the SON verification concept in other areas is of high interest, e.g., Wi-Fi networks.

11.4.3.1 Communication Handling

Verification areas, as given by Definition 5.3, will get larger and include more entities requiring assessment, which creates a potential for more area overlaps and verification collisions (cf. Definition 3.3). Therefore, the scope of the verification process will no longer solely include verification areas, but form so-called *verification collision domains* [TAT16]. A collision domain is a section of the mobile network where verification collisions have taken place, i.e., areas being in a verification collision are part of such a domain. Areas that are not participating in collisions form their own collision domain.

Furthermore, a verification mechanism will extend its capabilities, besides assessing verification areas and generating corrective undo or topology actions. The verification process needs a much more active interaction with SON/cognitive functions. Concretely, to involve functions into the corrective action decision making process, it would need to reliably negotiate all required verification parameters with all involved entities.

Figure 11.1 shows a potential design change [TATS16]. The communication flow is realized as a *three-way handshake*. Before executing a corrective action, the verification process contacts the functions of each collision domain over a verification interface by

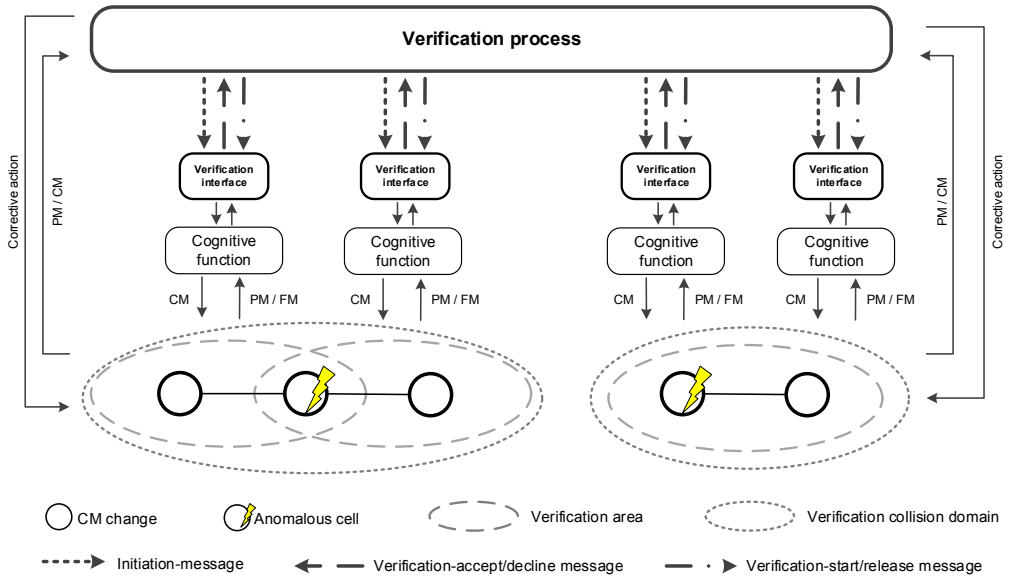


Figure 11.1: Verification collision domains and three-way handshake communication

sending an *initiation message*. It serves the purpose of notifying the functions within the verification areas about the spatial scope of the verification process. There are, however, two cases between which we would need to differentiate. The first one is when the verification mechanism notifies functions that are operating in a collision domain comprised of more than one verification area. Should this be the case, it will need to include not only the list of impacted cells and the change type, but also a parameter that specifies the time required for resolving the collisions.

Upon reception of such a message, each function will respond with either a *verification-accept* or with a *verification-decline* message. By sending an accept message, the replying function is declaring its decision not to interfere as corrective actions are getting executed within the collision domain. However, a function may also decline the planned corrective action if it needs additional steps to achieve its goal. Thereby, a function will be obligated to inform the verification process about the parameters it is optimizing as well as the estimated time for achieving its objective.

After receiving all replies, the verification process would send either a *verification-start* or a *verification-release* message to the functions within each collision domain. In case all functions have reported that they are willing to freeze their operation for the suggested time, a start message is sent, which will be later followed by the first set of corrective actions. However, if a function has reported that it is on the way of achieving its objective, the verification process will be obligated to notify the remaining functions within the same domain about the expected verification time.

Implementing those functionalities and studying the behavior of network is of high interest for future deployments.

11.4.3.2 Study of Algorithms

The second research area is the adaptation of the utilized algorithms. As introduced in Chapters 5, 6, and 7, the concept of SON verification makes use of numerous algorithms, in most cases such known from graph theory. For example, the topology verification process uses the Steiner tree algorithm to find the most appropriate corrective actions. However, it also utilizes several helper functions, as summarized in Section 10.6.1. They are crucial for the topology verification process as they affect its outcome. Hence, it becomes of particular interest to study the changes in those functions, e.g., adding prediction capabilities in the edge weighting function $d^{\mathcal{T}}$ (cf. Section 7.2).

Furthermore, it is of high importance to observe the behavior of the CM verification process in a much denser environment. Due to the fact that potentially more entities will require verification, the likelihood of verification collisions to emerge increases. Thus, one future research topic is the study of ability of the MST-based clustering algorithm to eliminate weak collisions, especially when a communication flow like the one presented in the previous section is implemented. Another would be the incorporation of new performance metrics into the priority rating functions ρ and $\tilde{\rho}$ required to resolve valid collisions. They are used by the constraint optimizer, as presented in Sections 6.5 and 6.6.

11.4.3.3 Other Application Areas

One of the most promising application areas is Wi-Fi SON (cf. Section 2.3.3.2). The concept of SON verification clearly fulfills the requirements for being a member of Wi-Fi SON. In its current state, it fits into the self-healing category since it is responsible for the assessment of configuration changes on the network performance and rolling back those harming it. This particular sequence is identified as CM verification. Also, the presented concept can be assigned to the self-managing category as it implements a topology verification process that switches elements on or off. The decision itself is based on the used utility function, which in the case of topology verification, is set to prevent overload from undertaking the performance of the network.

However, to fully operate in a Wi-Fi environment several questions need to be answered. First and foremost, the KPIs of interest as well as the profiling procedure have to be selected. In the latter case, research on the incomplete profile issue (cf. Definition 3.12) is necessary. The likelihood of such profiles to emerge must be determined as well as the entities that can be potentially affected need to be specified. Second, to enable CM verification it is crucial how we select function ζ from Definition 3.3, i.e., the condition for generating a rollback action. Third, it is of high importance to study the possibility of an over-constrained corrective action plan (cf. Definition 3.7) to emerge, in particular, how the upper time slot limit τ impacts the outcome of the CM verification process. Fourth, it is important to specify the edge removal function ξ used by the MST clustering technique (cf. Section 6.3). Wrong corrective actions can be executed if too many edges are removed from the cell behavior graph (cf. Definition 6.2). Fifth, to enable topology verification, function $d^{\mathcal{T}}$ that is used to compute the Steiner tree must be specified. In other words, the metric that is considered while forming the Steiner tree requires specification.

Beyond those research questions, it is of interest to apply the Steiner tree-based approach to solve other problems besides those already introduced in Chapter 7. It is not uncommon for troubleshooting approaches to incorporate several other features that improve the overall decision making process. For instance, in [BCL+09] a method for detecting the so-called rate anomaly problem in Wi-Fi networks is proposed. The problem is that stations with lower signal quality transmit at lower rates and, at the same time, consume a significant majority of airtime. Thereby, the throughput of stations transmitting at higher rates is significantly reduced. The authors suggest the inclusion of energy consumption metrics into their utility function to improve the outcome of their algorithm.

In the terms of the Steiner tree-based method, such an extension would be added to edge weighting function $d^{\mathcal{T}}$, as described in Section 7.2. Concretely, energy consumption measurements and metrics that estimate the resilience state of the network can be included. In the latter case, the algorithm may form a second Steiner tree, whose Steiner points represent backup network elements that are required to operate only in the case of faults and disruption of the normal operation.

Part VI

Appendix

Acronyms

3GPP	3rd Generation Partnership Project
ACK	Acknowledgment
AMPS	Advanced Mobile Phone System
ANR	Automatic Neighbor Relation
API	Application Programming Interface
ARP	Allocation Retention Priority
BLU	Blocked by User
CAAC	Cell Association Auto-Configuration
CBR	Constant Bit Rate
CCO	Coverage and Capacity Optimization
CIO	Cell Individual Offset
CKPI	Configuration management verification KPI
CM	Configuration Management
COC	Cell Outage Compensation
CQI	Channel Quality Indicator
CSP	Constraint Satisfaction Problem
CSSR	Call Setup Success Rate
CSV	Comma Separated Value
CVSI	Cell Verification State Indicator
DHCP	Dynamic Host Configuration Protocol
DM	Domain Management

DTP Decision-Theoretic Planning

E-RAB EUTRAN Radio Access Bearer

EDGE Enhanced Data Rates for GSM Evolution

EM Element Manager

eNB Evolved NodeB

EPC Evolved Packet Core

ESM Energy Saving Management

ETSI European Telecommunications Standards Institute

EUTRAN Evolved Universal Terrestrial Radio Access Network

FM Fault Management

FRGNG Fixed Resolution Growing Neural Gas

GBR Guaranteed Bit Rate

GERAN GSM EDGE Radio Access Network

GRAN GSM Radio Access Network

GNG Growing Neural Gas

GPRS General Packet Radio Service

GSM Global System for Mobile Communications

GUI Graphical User Interface

HSS Home Subscriber Server

NMT Nordic Mobile Telephone

HDP Hierarchical Dirichlet Process

HOSR Handover Success Rate

HSPA High Speed Packet Access

IEEE Institute of Electrical and Electronics Engineers

IMT International Mobile Telecommunications

IO Input-Output

IP Internet Protocol

ISDN Integrated Services Digital Network

JVM Java Virtual Machine

JWt Java Web Toolkit

KPI Key Performance Indicator

KQI Key Quality Indicator

LTE Long Term Evolution

MAC Medium Access Control

MIMO Multiple-Input and Multiple-Output

MLB Mobility Load Balancing

MLN Markov Logic Network

MME Mobility Management Entity

MRO Mobility Robustness Optimization

MST Minimum Spanning Tree

NACK Non-Acknowledgment

NE Network Element

NM Network Management

NRT Neighbor Relation Table

OAM Operation, Administration and Management

ODE Ordinary Differential Equation

OSS Operations Support System

PCI Physical Cell Identity

PDCP Packet Data Convergence Protocol

PDN Packet Data Network

PDU Protocol Data Unit

PLMN Public Land Mobile Network

PM Performance Management

PRB Physical Resource Block

QCI QoS Class Identifier

QoS Quality of Service

RACH Random Access Channel

RAN Radio Access Network

RAT Radio Access Technology

RET Remote Electrical Tilt

RLC Radio Link Control

RLF Radio Link Failure

RRC Radio Resource Control

RRM Radio Resource Management

RSRP Reference Signal Received Power

RSRQ Reference Signal Received Quality

RSS Received Signal Strength

S3 SON Simulation System

SGSN Serving GPRS Support Node

SIB System Information Block

SINR Signal to Interference plus Noise Ratio

SON Self-Organizing Network

SWMN Self-optimizing Wireless Mesh Network

TKPI Topology verification KPI

TS Technical Specification

TR Technical Report

TXP Transmission Power

UE User Equipment

UMTS Universal Mobile Telecommunications System

UTRAN Universal Terrestrial Radio Access Network

VoIP Voice over IP

WCDMA Wideband Code Division Multiple Access

WLAN Wireless Local Area Network

List of Symbols

Actions

$c^{\perp}$	A corrective undo (rollback) action. It restores a cell's configuration to a previous stable state and is the outcome of a process that verifies configuration changes (cf. Section 5.4.1).
$\mathcal{P}^{C^{\perp}} = \{\hat{C}_1^{\perp}, \ldots, \hat{C}_i^{\perp}\}$	A partition of the set of all corrective undo (rollback) actions $C^{\perp}$, also referred to as a corrective action plan (cf. Definition 3.4).
$C^{\perp}$	Represents the set of all corrective undo (rollback) actions $c^{\perp}$ that are generated by a verification process (cf. Section 5.4.1).
δ_i	Represents an action which is either a single CM change or a set of CM changes. An action is executed by a SON function.
$\Delta = \{\delta_1, \ldots, \delta_i\}$	A SON function transaction as given by Definition 3.2, i.e., a sequence of actions δ_i that are required to a reach the function's objective.
$\mathcal{G}(\hat{C}_i^{\perp})$	Gain of executing a block (step) $\hat{C}_i^{\perp}$ of a corrective action plan $\mathcal{P}^{C^{\perp}} = \{\hat{C}_1^{\perp}, \ldots, \hat{C}_i^{\perp}\}$ (cf. Definition 3.4). The gain is required for the specification of the gain-aware property (cf. Definition 3.6).
$c^{\vdash}$	A topology corrective action. It either enables or disables a cell. It is generated by the topology verification process (cf. Section 5.4.2).
$C^{\vdash}$	Set of all topology corrective actions that are generated by the topology verification process (cf. Section 5.4.2).

Counters

τ	Upper limit of the number of blocks of a corrective action plan $\mathcal{P}^{C^{\perp}} = \{\hat{C}_1^{\perp}, \ldots, \hat{C}_i^{\perp}\}$ (cf. Definition 3.4).
α	An update factor used for the computation of the Cell Verification State Indicator (CVSI) (cf. Equation 6.4), also referred to as the cell anomaly level.

κ	Number of cell clusters that emerge after triggering the MST clustering approach during the process of CM verification (cf. Section 6.3).
ν	Represents the number of neighbors that are considered by an on-demand cell for the computation of its TKPI anomaly level.
t	An index that represents a time interval, for example, a granularity period (cf. Section 2.2.3).

Functions

$\varphi\colon P^{\perp} \times K^{\perp} \to A^{\perp}$	CKPI anomaly level function (cf. Section 6.1). It is used by the CM verification process.
$\psi\colon A^{\perp} \to \mathbb{R}$	Vector aggregation function. It is used during the anomaly detection phase of the CM verification process (cf. Section 5.3).
$\tilde{\rho}\colon \tilde{\Theta} \to \mathbb{R}$	Priority function that rates a soft constraint during the process of solving an over-constrained verification collision problem (cf. Section 6.6).
$r\colon \tilde{\Theta} \to \{0, 1\}$	Reification function that takes a soft constraint and returns 0 or 1. It is used during the process of identifying soft verification collisions (cf. Section 6.6).
$\tilde{\omega}\colon V^{\Phi} \to \tilde{X}$	Variable assignment function. It is used during the process of finding and eliminating soft verification collisions (cf. Section 6.6).
$\omega\colon V^{\Phi} \to X$	Variable assignment function. It is used during the process of resolving verification collisions (cf. Section 6.5).
$\rho\colon X \to \mathbb{R}$	Priority function that rates a variable that represents an undo action (cf. Section 6.5).
$\vartheta\colon A^{\perp} \times \mathbb{R} \to \mathbb{R}$	The Cell Verification State Indicator (CVSI) (cf. Equation 6.4), also referred to as the cell anomaly level.
$m\colon V^{\Phi} \to \mathcal{C}$	A function that assigns each vertex of a verification collision graph $G^{\Phi} = (V^{\Phi}, E^{\Phi})$ a color from the set $\mathcal{C}$.
$d\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$	Distance function for forming the MST during the cell clustering procedure (cf. Section 6.3).
$\xi\colon T^{\Sigma} \to \mathbb{P}(\tilde{E}^{\Sigma})$	Edge removal function. It is required by the MST clustering procedure, as defined in Section 6.3.
$\zeta\colon C^{\perp} \to \mathbb{P}(\Sigma) \setminus \emptyset$	Function that identifies the origin, i.e., cells that triggered the generation of a rollback action.

$\widetilde{\zeta}\colon C^{\perp} \to \mathbb{P}(\Sigma^{A}) \setminus \emptyset$ A function that returns the anomalous cells that triggered the generation of an undo action.

$\dot{\varphi}\colon \dot{P}^{\vdash} \times \dot{K}^{\vdash} \to \dot{A}^{\vdash}$ TKPI anomaly level function. It is used to specify the cell behavior during topology verification (cf. Section 7.1).

$\iota\colon \mathring{\Sigma}^{\mathcal{R}} \times \mathring{K}^{\vdash} \to V^{\mathcal{P}} \cup \emptyset$ Steiner point conversion function. It is used during topology verification and has the purpose of converting a Steiner point to a terminal node (cf. Section 7.4).

Graphs

$\chi(G^{\Phi})$ The verification collision grade (cf. Definition 6.5), i.e., the Chromatic number of the verification graph $G^{\Phi} = (V^{\Phi}, E^{\Phi})$ (cf. Definition 6.4).

$\bar{V}^{\Phi}$ Represents a clique formed within the verification graph $G^{\Phi} = (V^{\Phi}, E^{\Phi})$ (cf. Section 6.6) during the process of finding soft verification collisions.

$\bigcup_{i=1}^{n} \bar{V}_{i}^{\Phi}$ The union of all cliques found during the process of solving an over-constrained verification collision problem.

$\check{G}^{\Phi} = (\check{V}^{\Phi}, \check{E}^{\Phi})$ Verification collision graph that is formed after the elimination of soft verification collisions (cf. Section 6.6).

$\check{V}^{\Phi}$ Set of vertexes for reduced verification graph $\check{G}^{\Phi}$.

$w(e^{\Sigma})$ Weight of an edge E^{Σ} in cell behavior graph $G^{\Sigma} = (V^{\Sigma}, E^{\Sigma})$, where $e^{\Sigma} \in E^{\Sigma}$ (cf. Definition 6.2).

F^{Σ} Represents a forest, i.e., undirected graph whose connected components are trees. Each tree represents a cell cluster that got formed after triggering the MST clustering procedure (cf. Section 6.3).

$G^{\Sigma} = (V^{\Sigma}, E^{\Sigma})$ Cell behavior graph (cf. Definition 6.2) formed during the process of CM verification.

$T^{\Sigma} = (\tilde{V}^{\Sigma}, \tilde{E}^{\Sigma})$ An MST formed out of the cell behavior graph G^{Σ} during the weak collision elimination procedure (cf. Section 6.3).

$\hat{T}^{\Sigma}$ Tree formed after removing edges from T^{Σ}. It appears during the MST clustering procedure (cf. Section 6.3).

$D_{G}(V^{\mathcal{F}})$ Complete graph of Steiner terminal vertexes. It is formed by the topology verification process (cf. Section 7.3).

$G^{\mathcal{T}} = (V^{\mathcal{T}}, E^{\mathcal{T}}, d^{\mathcal{T}})$ Steiner input graph (cf. Definition 7.3) formed by the topology verification process.

$V^{\mathcal{P}}$ Set of all steiner points. It is formed by the process of topology verification (cf. Section 7.3).

$V^{\mathcal{U}}$ Set of unnecessary Steiner points. It is formed by the process of topology verification (cf. Section 7.3).

$V^{\mathcal{R}}$ Set of required Steiner points. It is formed by the process of topology verification (cf. Section 7.3).

$V^{\mathcal{F}}$ Set of terminal vertexes in the Steiner tree. It is formed by the process of topology verification (cf. Section 7.3).

$T^{\mathcal{T}} = (\tilde{V}^{\mathcal{T}}, \tilde{E}^{\mathcal{T}})$ Steiner tree that is formed by the topology verification process, as given by Algorithm 2.

$\mathcal{C}$ Set of colors required to determine the verification collision grade, as given by Definition 6.5.

$G^{\Phi} = (V^{\Phi}, E^{\Phi})$ Verification collision graph, where the vertex set V^{Φ} represents verification areas and the set E^{Φ} verification collisions (cf. Definition 6.4).

KPI Types, Anomaly Levels, and Profiles

$k^{\perp}$ Represents a CKPI, as given by Definition 5.5. It is element of a CKPI vector $\mathbf{k}^{\perp} = (k_1^{\perp}, \ldots, k_n^{\perp})$. It is utilized by CM verification.

$k^{\vdash}$ Represents a TKPI, as given by Definition 5.6. It is element of a TKPI vector $\mathbf{k}^{\vdash} = (k_1^{\vdash}, \ldots, k_n^{\vdash})$ of a cell. It is utilized by topology verification.

$\dot{k}^{\vdash}$ Represents a TKPI of a static cell, as given by Definition 5.6. It is element of a TKPI vector $\dot{\mathbf{k}}^{\vdash} = (\dot{k}_1^{\vdash}, \ldots, \dot{k}_n^{\vdash})$.

$\mathring{k}^{\vdash}$ Represents a TKPI of an on-demand cell, as given by Definition 5.6. It is element of a TKPI vector $\mathring{\mathbf{k}}^{\vdash} = (\mathring{k}_1^{\vdash}, \ldots, \mathring{k}_n^{\vdash})$.

$a^{\perp}$ An anomaly level of a CKPI (cf. Definition 6.1). It is element of a CKPI anomaly level vector $\mathbf{a}^{\perp} = (a_1^{\perp}, \ldots, a_n^{\perp})$.

$p^{\perp}$ The profile of a CKPI (cf. Section 5.3.2). It is element of a CKPI profile vector $\mathbf{p}^{\perp} = (p_1^{\perp}, \ldots, p_n^{\perp})$.

$\dot{p}^{\vdash}$ The profile of a TKPI (cf. Section 5.3.3). It is element of a TKPI profile vector $\dot{\mathbf{p}}^{\vdash} = (\dot{p}_1^{\vdash}, \ldots, \dot{p}_n^{\vdash})$. Such profiles are defined for static cells only.

$a^{\vdash}$ A TKPI anomaly level of a cell. It is element of a TKPI anomaly level vector $\mathbf{a}^{\vdash} = (a_1^{\vdash}, \ldots, a_n^{\vdash})$, as given by Definition 7.1.

$\dot{a}^{\vdash}$ An anomaly level of TKPI of a static cell (cf. Definition 7.1). It is element of a TKPI anomaly level vector $\dot{\mathbf{a}}^{\vdash} = (\dot{a}_1^{\vdash}, \ldots, \dot{a}_n^{\vdash})$.

$\mathring{a}^{\vdash}$ An anomaly level of TKPI of an on-demand cell (cf. Definition 7.1). It is element of a TKPI anomaly level vector $\mathring{\mathbf{a}}^{\vdash} = (\mathring{a}_1^{\vdash}, \ldots, \mathring{a}_n^{\vdash})$.

Common Mathematical Symbols

B_n Bell number, i.e., the number of partitions of a set of size n.

$\emptyset$ The empty set.

$\mathbb{N}^+$ Set of positive natural numbers excluding 0.

$\mathbb{N}_0$ Set of positive natural numbers including 0.

$\mathbb{N}$ Set of natural numbers.

$\mathcal{P}^S, \mathcal{P}(S)$ Partition of a set S.

$\mathbb{P}(S)$ Power set of a set S.

$\mathbb{R}^n$ Real coordinate space of n dimensions.

$\mathbb{R}$ Set of real numbers.

$\mathbb{R}_{\geq 0}$ Set of positive real numbers including 0.

Constraint Optimization

Θ The set of all hard constraints, i.e., constraints that must not be violated. They are used during the process of generating a corrective action plan (cf. Section 6.5).

$\tilde{\Theta}$ The set of all soft constraints, i.e., constraints that can be violated under certain circumstances. They are used during the process of solving an over-constrained verification collision problem (cf. Section 6.6).

$\tilde{x}$ A variable which represents an edge $e^{\Phi} \in E^{\Phi}$ of the verification collision graph $G^{\Phi} = (V^{\Phi}, E^{\Phi})$. Such a variable is used during the process of solving an over-constrained verification collision problem (cf. Section 6.6).

$\tilde{X}$ The set of all variables $\tilde{x}$. They are used during the process of solving an over-constrained verification collision problem (cf. Section 6.6).

x A variable which represents a vertex $v^{\Phi} \in V^{\Phi}$ of the verification collision graph $G^{\Phi} = (V^{\Phi}, E^{\Phi})$. Variables are required during the verification collision resolving process (cf. Section 6.5).

X The set of all verification variables x. Variables are required during the verification collision resolving process (cf. Section 6.5).

Cell Sets and Areas

Σ^{M} The set of all cells that are monitored by the verification process. It is formed as the union of all cells of all verification areas (cf. Section 5.2).

Σ^{A} Set of all anomalous cells in the network. Note that $\Sigma^{A} \subseteq \Sigma$, where Σ is the set of all cells.

Σ Set of all cells in the network. It includes both static and on-demand cells (cf. Section 5.2.2).

$\dot{\Sigma}$ Set of all static cells, i.e., cells that are never turned off during their operation. Such cells are required by the Steiner tree-based verification algorithm (cf. Chapter 7).

$\mathring{\Sigma}$ Set of all on-demand cells, i.e., cells that can be switched on or off during their operation. Such cells are required by the Steiner tree-based verification algorithm (cf. Chapter 7).

$\Sigma^{M'}$ Set of cells that represent a verification area (cf. Section 5.2). Note that $\Sigma^{M'} \subseteq \Sigma^{M}$, where Σ^{M} is the set of all cells monitored by the verification process.

φ Represents a verification area (cf. Definition 5.3), i.e., a set of cells that are assessed by the verification process.

Φ Set of all verification areas that are formed by the verification process.

Cell Types

σ^{A} An anomalous cell detected by the verification algorithm. Note that the set of all anomalous cells is denoted as Σ^{A}.

σ Cell in the network. Note that the set of all cells is denoted as Σ.

$\dot{\sigma}$ A static cell. Such a cell is required by the Steiner tree-based verification algorithm (cf. Chapter 7).

$\mathring{\sigma}$	An on-demand cell. Such a cell is required by the Steiner tree-based verification algorithm (cf. Chapter 7).

Vectors and Vector Spaces

$\mathbf{k}^{\perp} = (k_1^{\perp}, \ldots, k_n^{\perp})$	A vector of CKPIs, as given by Definition 5.5. CKPIs are relevant for the CM verification process (cf. Chapter 6).
$K^{\perp} = \{\mathbf{k}_1^{\perp}, \ldots, \mathbf{k}_{\lvert\Sigma\rvert}^{\perp}\}$	The CKPI vector space, as given by Definition 5.5. It is relevant for the CM verification process (cf. Chapter 6).
$\mathbf{k}^{\vdash} = (k_1^{\vdash}, \ldots, k_n^{\vdash})$	A vector of TKPIs, as given by Definition 5.6. The vector is relevant for the topology verification process (cf. Chapter 7).
$K^{\vdash} = \{\mathbf{k}_1^{\vdash}, \ldots, \mathbf{k}_{\lvert\Sigma\rvert}^{\vdash}\}$	The TKPI vector space, as given by Definition 5.6. It is relevant for the topology verification process (cf. Chapter 7). Note that it is further divided into the TKPI vector space of all static and on-demand cells, denoted as $\dot{K}^{\vdash} = \{\dot{\mathbf{k}}_1^{\vdash}, \ldots, \dot{\mathbf{k}}_{\lvert\dot{\Sigma}\rvert}^{\vdash}\}$ and $\mathring{K}^{\vdash} = \{\mathring{\mathbf{k}}_1^{\vdash}, \ldots, \mathring{\mathbf{k}}_{\lvert\mathring{\Sigma}\rvert}^{\vdash}\}$, respectively.
$\mathbf{a}^{\perp} = (a_1^{\perp}, \ldots, a_n^{\perp})$	A CKPI anomaly level vector (cf. Definition 6.1). It is required to define the state of a cell with respect to CM verification.
$A^{\perp} = \{\mathbf{a}_1^{\perp}, \ldots, \mathbf{a}_{\lvert\Sigma\rvert}^{\perp}\}$	The CKPI anomaly level vector space, as given by Definition 6.1. It is relevant for the CM verification process.
$\mathbf{p}^{\perp} = (p_1^{\perp}, \ldots, p_n^{\perp})$	A profile vector that defines the expected behavior of the CKPIs of a cell (cf. Section 5.3.2).
$P^{\perp}$	Profile vector space for the given CKPI profile vectors, as introduced in Section 5.3.2.
$\dot{\mathbf{p}}^{\vdash} = (\dot{p}_1^{\vdash}, \ldots, \dot{p}_n^{\vdash})$	A profile vector that defines the expected behavior of the TKPIs of a static cell (cf. Section 5.3.3).
$\dot{P}^{\vdash}$	Profile vector space for the given TKPI profile vectors (cf. Section 5.3.3).
$\mathbf{a}^{\vdash} = (a_1^{\vdash}, \ldots, a_n^{\vdash})$	A TKPI anomaly level vector. The elements represent TKPIs anomaly levels (cf. Definition 7.1) which are required to define the state of a cell with respect to topology verification. It should be noted that the anomaly level vectors of static and on-demand cells are denoted as $\dot{\mathbf{a}}^{\vdash} = (\dot{a}_1^{\vdash}, \ldots, \dot{a}_n^{\vdash})$ and $\mathring{\mathbf{a}}^{\vdash} = (\mathring{a}_1^{\vdash}, \ldots, \mathring{a}_n^{\vdash})$, respectively.
$A^{\vdash} = \{\mathbf{a}_1^{\vdash}, \ldots, \mathbf{a}_{\lvert\Sigma\rvert}^{\vdash}\}$	The TKPI anomaly level vector space (cf. Definition 7.2). Note that the anomaly level vector space of static and on-demand cells are denoted as $\dot{A}^{\vdash} = \{\dot{\mathbf{a}}_1^{\vdash}, \ldots, \dot{\mathbf{a}}_{\lvert\dot{\Sigma}\rvert}^{\vdash}\}$ and $\mathring{A}^{\vdash} = \{\mathring{\mathbf{a}}_1^{\vdash}, \ldots, \mathring{\mathbf{a}}_{\lvert\mathring{\Sigma}\rvert}^{\vdash}\}$, respectively.

List of Figures

List of Tables

List of Listings

List of Definitions

Bibliography

[3GP98] 3GPP. Universal Mobile Telecommunications System (UMTS); Selection procedures for the choice of radio transmission technologies of the UMTS (UMTS 30.03 version 3.2.0). Technical report tr 101 112 v3.2.0, 3rd Generation Partnership Project (3GPP), April 1998.

[3GP01] 3GPP. Telecommunication management; Configuration Management (CM); Part 1: Concept and requirements. Technical specification 32.106-1 v4, 3rd Generation Partnership Project (3GPP), March 2001.

[3GP10] 3GPP. Telecommunication management; Study on Energy Savings Management (ESM). Technical specification 32.826 v10.0.0, 3rd Generation Partnership Project (3GPP), April 2010.

[3GP12] 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification. Technical specification 36.331 v10, 3rd Generation Partnership Project (3GPP), March 2012.

[3GP13a] 3GPP. Telecommunication management; Fault Management; Part 1: 3G fault management requirements. Technical specification 32.111-1 v12.0.0, 3rd Generation Partnership Project (3GPP), June 2013.

[3GP13b] 3GPP. Telecommunication management; Self-Organizing Networks (SON) Policy Network Resource Model (NRM) Integration Reference Point (IRP); Information Service (IS). Technical specification 32.522 v11.7.0, 3rd Generation Partnership Project (3GPP), September 2013.

[3GP14a] 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures. Technical specification 36.213 v12.1.0, 3rd Generation Partnership Project (3GPP), March 2014.

[3GP14b] 3GPP. Telecommunication management; Study on Network Management (NM) centralized Coverage and Capacity Optimization (CCO) Self-Organizing Networks (SON) function. Technical specification 32.836 v12.0.0, 3rd Generation Partnership Project (3GPP), September 2014.

[3GP16a] 3GPP. 3GPP Specifications Groups and TSG Structure. `http://www.3gpp.org/specifications-groups`, 2016. Visited: October 2016.

[3GP16b] 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2. Technical specification 36.300 v13.2, 3rd Generation Partnership Project (3GPP), January 2016.

[3GP16c] 3GPP. Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access Network (E-UTRAN); S1 Application Protocol (S1AP). Technical specification 36.413 v14.0.0, 3rd Generation Partnership Project (3GPP), September 2016.

[3GP16d] 3GPP. Technical Specification Group Services and System Aspects; General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access. Technical specification 23.401 v14.2.0, 3rd Generation Partnership Project (3GPP), December 2016.

[3GP16e] 3GPP. Technical Specification Group Services and System Aspects; Telecommunication management;Performance Management (PM);. Technical specification 32.401 v13.1.0, 3rd Generation Partnership Project (3GPP), June 2016.

[3GP16f] 3GPP. Telecommunication management; Key Performance Indicators (KPI) for Evolved Universal Terrestrial Radio Access Network (E-UTRAN): Definitions. Technical specification 32.450 v13, 3rd Generation Partnership Project (3GPP), January 2016.

[3GP16g] 3GPP. Telecommunication Management; Self-Organizing Networks (SON); Self-healing Concepts and Requirements. Technical specification 32.541 v13, 3rd Generation Partnership Project (3GPP), January 2016.

[4GA11] 4GAmericas. Self-Optimizing Networks: Benefits of SON in LTE, July 2011. White paper.

[AFG+08] Mehdi Amirijoo, Pål Frenger, Fredrik Gunnarsson, Harald Kallin, Johan Moe, and Kristina Zetterberg. Neighbor Cell Relation List and Physical Cell Identity Self-Organization in LTE. In *ICC Workshops - 2008 IEEE International Conference on Communications Workshops*, pages 37–41, Beijing, PRC, May 2008.

[AJLS11] Mehdi Amirijoo, Ljupco Jorguseski, Remco Litjens, and Lars Christoph Schmelz. Cell Outage Compensation in LTE Networks: Algorithms and Performance Assessment. In *IEEE Vehicular Technology Conference (VTC Spring 2011)*, pages 1–5, Budapest, Hungary, May 2011.

[All17] Wi-Fi Alliance. Wi-Fi Alliance publishes 7 for '17 Wi-Fi predictions. `https://wi-fi.org/news-events/newsroom/`

wi-fi-alliance-publishes-7-for-17-wi-fi-predictions, January 2017. Visited: January 2017.

[ATT16a] Janne Ali-Tolppa and Tsvetko Tsvetkov. Network Element Stability Aware Method for Verifying Configuration Changes in Mobile Communication Networks. In *IFIP Autonomous Infrastructure, Management and Security (AIMS 2016)*, Munich, Germany, June 2016.

[ATT16b] Janne Ali-Tolppa and Tsvetko Tsvetkov. Optimistic Concurrency Control in Self-Organizing Networks Using Automatic Coordination and Verification. In *IEEE/IFIP Network Operations and Management Symposium (NOMS 2016)*, Istanbul, Turkey, April 2016.

[Ban13] Tobias Bandh. *Coordination of Autonomic Function Execution in Self-Organizing Networks*. Phd thesis, Technische Universität München, April 2013. ISBN 3-937201-34-3.

[BCL+09] Paramvir Bahl, Ranveer Chandra, Patrick P. C. Lee, Vishal Misra, Jitendra Padhye, Dan Rubenstein, and Yan Yu. Opportunistic Use of Client Repeaters to Improve Performance of WLANs. *IEEE/ACM Transactions on Networking*, 17(4):1160–1171, August 2009.

[BCS11] Sueng Jae Bae, Min Young Chung, and Jungmin So. Handover triggering mechanism based on IEEE 802.21 in heterogeneous networks with LTE and WLAN. In *The International Conference on Information Networking (ICOIN 2011)*, pages 399–403, January 2011.

[BDH99] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[BFZ+14] Sascha Berger, Albrecht Fehske, Paolo Zanier, Ingo Viering, and Gerhard Fettweis. Comparing Online and Offline SON Solutions for Concurrent Capacity and Coverage Optimization. In *Proceedings of the Vehicular Technology Conference (VTC Fall 2014)*, Vancouver, Canada, September 2014.

[BHO16] Anne-Marie Bosneag, Sidath Handurukande, and James O'Sullivan. Automatic Discovery of Sub-Optimal Radio Performance in LTE RAN Networks. In *IEEE/IFIP Network Operations and Management Symposium (NOMS 2016)*, Istanbul, Turkey, April 2016.

[Bie14] Adam Bien. Structuring Complex JavaFX 8 Applications for Productivity, October 2014.

[BKKS16] Levente Bodrog, Márton Kajó, Szilárd Kocsis, and Benedek Schultz. A Robust Algorithm for Anomaly Detection in Mobile Networks. In *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2016)*, Valencia, Spain, September 2016.

[BM82] John Adrian Bondy and U. S. R. Murty. *Graph Theory With Applications*. Elsevier Science Ltd, fifth printing edition, 1982. ISBN: 0-444-19451-7.

[BMQS08] Javier Baliosian, Katarina Matusikova, Karl Quinn, and Rolf Stadler. Policy-based Self-healing for Radio Access Networks. In *IEEE/IFIP Network Operations and Management Symposium (NOMS 2008)*, Salvador, Brazil, April 2008.

[BRS+10] Tobias Bandh, Raphael Romeikat, Henning Sanneck, Lars Christoph Schmelz, Bernhard Bauer, and Georg Carle. Optimized Network Configuration Parameter Assignment Based on Graph Coloring. In *Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS 2010)*, Osaka, Japan, April 2010.

[BRS11] Tobias Bandh, Raphael Romeikat, and Henning Sanneck. Policy-Based Coordination and Management of SON Functions. In *IFIP/IEEE International Symposium on Integrated Network Management (IM 2011)*, pages 827–840, Dublin, Ireland, May 2011.

[Bru09] Richard A. Brualdi. *Introductory Combinatorics*. Pearson Prentice Hall, fifth edition, 2009. ISBN 978-0-13-602040-0.

[CAA13] Richard Combes, Zwi Altman, and Eitan Altman. Coordination of Autonomic Functionalities in Communications Networks. In *11th International Symposium on Modeling & Optimization in Mobile, Ad Hoc & Wireless Networks (WiOpt 2013)*, Tsukuba, Japan, May 2013.

[CCC+14a] Gabriela Ciocarlie, Chih-Chieh Cheng, Christopher Connolly, Ulf Lindqvist, et al. Managing Scope Changes for Cellular Network-level Anomaly Detection. In *International Workshop on Self-Organizing Networks (IWSON 2014)*, Barcelona, Spain, August 2014.

[CCC+14b] Gabriela Ciocarlie, Christopher Connolly, Chih-Chieh Cheng, Ulf Lindqvist, et al. Anomaly Detection and Diagnosis for Automatic Radio Network Verification. In *6th International Conference on Mobile Networks and Management (MONAMI 2014)*, Würzburg, Germany, September 2014.

[CGT+11] Marinos Charalambides, Alex Galis, Daphne Tuncer, Stuart Clayman, Stylianos Georgoulas, et al. Deliverable D3.4 Cooperation Strategies and Incentives. Technical report, UniverSelf Project, July 2011.

[CLN+14] Gabriela Ciocarlie, Ulf Lindqvist, Kenneth Nitz, Szabolcs Nováczki, and Henning Sanneck. On the Feasibility of Deploying Cell Anomaly Detection in Operational Cellular Networks. In *IEEE/IFIP Network Operations and Management Symposium (NOMS 2014)*, Krakow, Poland, May 2014.

[CLRS09] Thomas H. Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction To Algorithms*. MIT Press, third edition, 2009. ISBN 978-0-262-03384-8.

[CSW+15] D. Chen, J. Schuler, P. Wainio, J. Salmelin, Pekka Wainio, and Juha Salmelin. 5G Self-Optimizing Wireless Mesh Backhaul. In *IEEE Conference on Computer Communications Workshops (INFOCOM WORKSHOPS 2015)*, pages 23–24, April 2015.

[CZ15] Shanzhi Chen and Jian Zhao. The Requirements, Challenges, and Technologies for 5G of Terrestrial Mobile Telecommunication. *Communications Magazine*, January 2015.

[DD09] Michel Marie Deza and Elena Deza. *Encyclopedia of Distances*. Springer Science & Business Media, 2009. ISBN: 978-3-642-00234-2.

[Dij59] Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271, December 1959.

[DJG+11] Anders Dahlén, Arne Johansson, Fredrik Gunnarsson, Johan Moe, Thomas Rimhagen, and Harald Kallin. Evaluations of LTE Automatic Neighbor Relations. In *IEEE Vehicular Technology Conference (VTC Spring 2011)*, pages 1–5, May 2011.

[EGK+04] John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull. *Graphviz and Dynagraph - Static and Dynamic Graph Drawing Tools*, pages 127–148. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[Eri12] Ericsson. Transparent Network-Performance Verification For LTE Rollouts. White Paper, 284 23-3179 Uen, September 2012.

[Eri14] Ericsson. 5G: what is it? White paper, October 2014.

[Fou15] Apache Software Foundation. *Apache Log4j 2, version 2.6*, May 2015.

[Fou16] Apache Software Foundation. Apache Commons. `https://commons.apache.org/`, 2016. Visited: June 2016.

[FPP07] David Freedman, Robert Pisani, and Roger Purves. *Statistics*. International student edition. W.W. Norton & Company, 2007.

[Fri95] Bernd Fritzke. A Growing Neural Gas Network Learns Topologies. In *Advances in Neural Information Processing Systems*, pages 625–632. MIT Press, 1995.

[FS09] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, June 2009. ISBN: 978-0-521-89806-5.

[FTS+14a] Christoph Frenzel, Tsvetko Tsvetkov, Henning Sanneck, Bernhard Bauer, and Georg Carle. Detection and Resolution of Ineffective Function Behavior in Self-Organizing Networks. In *IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM 2014)*, Sydney, Australia, June 2014.

[FTS+14b] Christoph Frenzel, Tsvetko Tsvetkov, Henning Sanneck, Bernhard Bauer, and Georg Carle. Operational Troubleshooting-enabled Coordination in Self-Organizing Networks. In *6th International Conference on Mobile Networks and Management (MONAMI 2014)*, Würzburg, Germany, September 2014.

[GAMK+16] Ana Gómez-Andrades, Pablo Muñoz, Emil J. Khatib, Isabel de la Bandera Cascales, Inmaculada Serrano, and Raquel Barco. Methodology for the Design and Evaluation of Self-Healing LTE Networks. *IEEE Transactions on Vehicular Technology*, 65(8):6468–6486, August 2016.

[GHNP01] Clemens Gröpl, Stefan Hougardy, Till Nierhoff, and Hans Jürgen Prömel. Lower Bounds for Approximation Algorithms for the Steiner Tree Problem. In *Lecture Notes in Computer Science*, pages 217–228, 2001.

[GJCT04] Alexander Gerdenitsch, Stefan Jakl, Yee Yang Chong, and Martin Toeltsch. A Rule-Based Algorithm for Common Pilot Channel and Antenna Tilt Optimization in UMTS FDD Networks. *Etri Journal*, 26(5):437–442, October 2004.

[GJS+14] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, and Alex Buckley. *The Java Language Specification, Java SE 8 Edition*. Addison-Wesley Professional, first edition, May 2014. ISBN: 978-0-13-390069-9.

[GKN15] Emden R. Gansner, Eleftherios Koutsofios, and Stephen North. *Drawing graphs with dot*, January 2015.

[GNM15] Borislava Gajic, Szabolcs Nováczki, and Stephen S. Mwanje. An Improved Anomaly Detection in Mobile Networks by Using Incremental Time-aware Clustering. In *IFIP/IEEE Workshop on Cognitive Network and Service Management (CogMan 2015)*, Ottawa, Canada, May 2015.

[Goo15] Google. Google Guava Library. `https://github.com/google/guava`, 2015. Visited: December 2015.

[Goo16] Google. Google Gson Library. `https://github.com/google/gson`, 2016. Visited: January 2016.

[GY05] Jonathan L. Gross and Jay Yellen. *Graph Theory and Its Applications*. Textbooks in Mathematics. Taylor & Francis, second edition, September 2005. ISBN: 978-1-584-88505-4.

[HBR95] David Heckerman, John S. Breese, and Koos Rommelse. Decision-Theoretic Troubleshooting. *Communications of the ACM*, 38:49–57, March 1995.

[HSS11] Seppo Hämäläinen, Henning Sanneck, and Cinzia Sartori, editors. *LTE Self-Organising Networks (SON): Network Management Automation for Operational Efficiency*. John Wiley & Sons, Chichester, UK, December 2011. ISBN: 978-1-119-97067-5.

[HT09] Harri Holma and Antti Toskala. *LTE for UMTS - OFDMA and SC-FDMA Based Radio Access*. Wiley Publishing, Chichester, UK, 2009. ISBN: 978-0-470-99401-6.

[HT12] Harri Holma and Antti Toskala. *LTE Advanced: 3GPP Solution for IMT-Advanced*. Wiley Publishing, Chichester, UK, 2012. ISBN 978-1-119-97405-5.

[IEE11] IEEE Computer Society. IEEE Guide - Adoption of the Project Management Institute (PMI®) Standard. A Guide to the Project Management Body of Knowledge (PMBOK®Guide) - Fourth Edition, November 2011. ISBN: 978-0-7381-6817-3.

[IEE16] IEEE Standards Association. IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pages 1–3534, December 2016.

[ISJB14] Ovidiu Iacoboaiea, Berna Sayrac, Sana Ben Jemaa, and Pascal Bianchi. SON Coordination for Parameter Conflict Resolution: A Reinforcement Learning Framework. In *IEEE Wireless Communications and Networking Conference (WCNC 2014)*, April 2014.

[JFG+13] Sana Ben Jemaa, Christoph Frenzel, Dario Götz, Beatriz González, Sören Hahn, et al. D5.1 Integrated SON Management Requirements and Basic Concepts. Technical report, SEMAFOUR Project, December 2013.

[JGr16] JGraph Ltd. *JGraphX (JGraph 6) User Manual, Version 3.6.0.0*, September 2016.

[JN09] Prasanta K. Jana and Azad Naik. An Efficient Minimum Spanning Tree based Clustering Algorithm. In *International Conference on Methods and Models in Computer Science*, pages 1–5, Delhi, India, December 2009.

[KAB+10] Thomas Kürner, Mehdi Amirijoo, Irina Balan, Hans van den Berg, Andreas Eisenblätter, et al. Final Report on Self-Organisation and its Implications in Wireless Access Networks. Deliverable d5.9, Self-Optimisation and self-ConfiguRATion in wirelEss networkS (SOCRATES), January 2010.

[Kim16] Michael Kimberlin. Reducing Boilerplate Code with Project Lombok. `http://jnb.ociweb.com/jnb/jnbJan2010.html`, 2016. Visited: September 2016.

[KMB81] Lawrence T. Kou, George Markowsky, and Leonard Berman. A Fast Algorithm for Steiner Trees. *Acta Informatica*, 15(2):141–145, June 1981.

[Kru56] Joseph B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.

[LAB+13] Daniela Laselva, Zwi Altman, Irina Balan, Andreas Bergström, Relja Djapic, et al. D4.1 SON Functions for Multi-Layer LTE and Multi-RAT Networks. Technical report, SEMAFOUR Project, November 2013.

[Lei79] Frank Thomson Leighton. A Graph Coloring Algorithm for Large Scheduling Problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–506, November 1979.

[Lig16] Lightbend. Configuration Library for JVM Languages. `https://github.com/typesafehub/config`, 2016. Visited: September 2016.

[LL08] Pierre Lescuyer and Thierry Lucidarme. *Evolved Packet System (EPS): The LTE and SAE Evolution of 3G UMTS*. Wiley Publishing, February 2008. ISBN: 978-0-470-05976-0.

[LLY+13] Kyunghan Lee, Joohyun Lee, Yung Yi, Injong Rhee, and Song Chong. Mobile Data Offloading: How Much Can WiFi Deliver? *IEEE/ACM Transactions on Networking*, 21(2):536–550, April 2013.

[LPCS04] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In *The First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, pages 15–28, 2004.

[LRL+05] Jaana Laiho, Kimmo Raivio, Pasi Lehtimäki, Kimmo Hätönen, and Olli Simula. Advanced Analysis Methods for3G Cellular Networks. *IEEE Transactions on Wireless Communications*, 4(3):930–942, May 2005.

[LSH16] Simon Lohmüller, Lars Christoph Schmelz, and Sören Hahn. Adaptive SON Management Using KPI Measurements. In *IEEE/IFIP Network Operations and Management Symposium (NOMS 2016)*, pages 625–631, Istanbul, Turkey, April 2016.

[Mar04] Dániel Marx. Graph coloring problems and their applications in scheduling. *Periodica Polytechnica Ser. El. Eng.*, 48:11–16, 2004.

[MAT16] Stephen S. Mwanje and Janne Ali-Tolppa. Fluid Capacity for Energy Saving Management in Multi-Layer Ultra-Dense 4G/5G Cellular Networks. In *International Conference on Network and Service Management (CNSM 2016)*, Montreal, Canada, November 2016.

[MATTS15] Stephen S. Mwanje, Janne Ali-Tolppa, Tsvetko Tsvetkov, and Henning Sanneck. A Framework for Cell-Association Auto Configuration of Network Functions in Cellular Networks. In *7th International Conference on Mobile Networks and Management (MONAMI 2015)*, Santander, Spain, September 2015.

[MBE11] Christian M. Mueller, Hajo Bakker, and Lutz Ewe. Evaluation of the Automatic Neighbor Relation Function in a Dense Urban Scenario. In *IEEE Vehicular Technology Conference (VTC Spring)*, pages 1–5, Yokohama, Japan, May 2011.

[MDM+16] Stephen S. Mwanje, Guillaume Decarreau, Christian Mannweiler, et al. Network Management Automation in 5G: Challenges and Opportunities. In *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2016)*, Valencia, Spain, September 2016.

[MSES12] Gilbert Micallef, Louai Saker, Salah E. Elayoubi, and Hans-Otto Scheck. Realistic Energy Saving Potential of Sleep Mode for Existing and Future Mobile Networks. *Journal of Communication*, 7(10):740–748, October 2012.

[NC16] Barak Naveh and Contributors. JGraphT: a free Java graph library. `http://jgrapht.org/`, 2016. Visited: September 2016.

[Nex15] Next Generation Mobile Networks Alliance. A Deliverable by the NGMN Alliance: NGMN 5G White Paper, February 2015. Final deliverable, version 1.0.

[NG15] Szabolcs Nováczki and Borislava Gajic. *Engineering Applications of Neural Networks: 16th International Conference, EANN 2015*, chapter Fixed-Resolution Growing Neural Gas for Clustering the Mobile Networks Data, pages 181–191. Springer International Publishing, September 2015.

[NGN08a] NGNM Alliance. A Deliverable by the NGMN Alliance NGMN Use Cases related to Self Organising Network, Overall Description, December 2008. Version 2.02.

[NGN08b] NGNM Alliance. Next Generation Mobile Networks Recommendation on SON and O&M Requirements, December 2008. Version 1.23.

[Nov13] Szabolcs Nováczki. An Improved Anomaly Detection and Diagnosis Framework for Mobile Network Operators. In *9th International Conference on Design of Reliable Communication Networks (DRCN 2013)*, March 2013.

[NPS11] M. Danish Nisar, Volker Pauli, and Eiko Seidel. Multi-RAT Traffic Steering – Why, when, and how could it be beneficial?, December 2011. White paper.

[NSN09] NSN. Self-Organizing Network (SON): Introducing the Nokia Siemens Networks SON Suite - an efficient, future-proof platform for SON. White Paper, October 2009.

[NTB+07] Gabor Nemeth, Peter Tarjan, Gergely Biczok, Ferenc Kubinszky, and Andras Veres. Measuring High-Speed TCP Performance During Mobile Handovers. In *IEEE Conference on Local Computer Networks (LCN 2007)*, pages 599–612, October 2007.

[NTS16] Szabolcs Nováczki, Tsvetko Tsvetkov, and Henning Sanneck. Scoring method and system for robust verification of configuration actions, WO patent application, PCT/EP2014/069096, 2016.

[NTSM15] Szabolcs Nováczki, Tsvetko Tsvetkov, Henning Sanneck, and Stephen Mwanje. A Scoring Method for the Verification of Configuration Changes in Self-Organizing Networks. In *7th International Conference on Mobile Networks and Management (MONAMI 2015)*, Santander, Spain, September 2015.

[Ora16] Oracle. JavaFX Frequently Asked Questions. `http://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html`, 2016. Visited: September 2016.

[Oxf05] The Oxford Dictionary of English. Revised Edition, Oxford University Press, 2005.

[PB98] Claude Le Pape and Philippe Baptiste. Resource Constraints for Preemptive Job-shop Scheduling. *Constraints*, 3(4):263 – 287, October 1998.

[PFL15] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2015.

[PKVB11] Kandaraj Piamrat, Adlen Ksentini, César Viho, and Jean-Marie Bonnin. QoE-aware Vertical Handover in Wireless Heterogeneous Networks. In *7th International Wireless Communications and Mobile Computing Conference (IWCMC 2011)*, pages 95–100, Istanbul, Turkey, July 2011.

[Qua16] Qualcomm. Qualcomm Wi-Fi SON. `https://www.qualcomm.com/products/features/wi-fi-son`, January 2016. Visited: December 2016.

[RH12] Juan Ramiro and Khalid Hamied. *Self-Organizing Networks (SON): Self-Planning, Self-Optimization and Self-Healing for GSM, UMTS and LTE*. Wiley Publishing, 1st edition, 2012. ISBN: 978-0-470-97352-3.

[RN10] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition, February 2010. ISBN: 978-0136042594.

[RPM05] J. Riihijarvi, M. Petrova, and P. Mahonen. Frequency allocation for WLANs using graph colouring techniques. In *Wireless On-demand Network Systems and Services, 2005 (WONS 2005)*, pages 216–222, St. Moritz, Switzerland, January 2005.

[RSB13] Raphael Romeikat, Henning Sanneck, and Tobias Bandh. Efficient , Dynamic Coordination of Request Batches in C-SON Systems. In *IEEE Veh. Technol. Conf. (VTC Spring 2013)*, Dresden, Germany, June 2013.

[RvBW06] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*. Elsevier Science, October 2006. ISBN: 9780444527264.

[SBS12] Péter Szilágyi, Tobias Bandh, and Henning Sanneck. Physical Cell ID Allocation in Multi-layer, Multi-vendor LTE Networks. In *4th International Conference on Mobile Networks and Management (MONAMI 2012)*, Hamburg, Germany, September 2012.

[SBT10a] Henning Sanneck, Yves Bouwen, and Eddy Troch. Context Based Configuration Management of Plug & Play LTE Base Stations. In *IEEE Network Operations and Management Symposium (NOMS 2010)*, pages 946–949, April 2010.

[SBT10b] Henning Sanneck, Yves Bouwen, and Eddy Troch. Dynamic Radio Configuration of Self-Organizing Base Stations. In *7th International Symposium on Wireless Communication Systems (ISWCS 2010)*, September 2010.

[Sch03] Jochen H. Schiller. *Mobile Communications*. Addison-Wesley, second edition, 2003. ISBN: 0-321-12381-6.

[SN12] Péter Szilágyi and Szabolcs Nováczki. An Automatic Detection and Diagnosis Framework for Mobile Communication Systems. *IEEE Transactions on Network and Service Management (TNSM)*, 9(2):184–197, June 2012.

[STB11] Stefania Sesia, Issam Toufik, and Matthew Baker, editors. *LTE - The UMTS Long Term Evolution: From Theory to Practice*. Wiley, second edition, July 2011. ISBN: 978-0-470-66025-6.

[STN15] Henning Sanneck, Tsvetko Tsvetkov, and Szabolcs Nováczki. Verification in Self-Organizing Networks, WO patent application, PCT/EP2014/058837, 2015.

[TAT16] Tsvetko Tsvetkov and Janne Ali-Tolppa. An Adaptive Observation Window for Verifying Configuration Changes in Self-Organizing Networks. In *Innovations in Clouds, Internet and Networks (ICIN 2016)*, Paris, France, March 2016.

[TATS16] Tsvetko Tsvetkov, Janne Ali-Tolppa, and Henning Sanneck. Method of verifying an operation of a mobile radio communication network, WO patent application, PCT/EP2015/055786, 2016.

[TATSC16a] Tsvetko Tsvetkov, Janne Ali-Tolppa, Henning Sanneck, and Georg Carle. A Minimum Spanning Tree-Based Approach for Reducing Verification Collisions in Self-Organizing Networks. In *IEEE/IFIP Network Operations and Management Symposium (NOMS 2016)*, Istanbul, Turkey, April 2016.

[TATSC16b] Tsvetko Tsvetkov, Janne Ali-Tolppa, Henning Sanneck, and Georg Carle. A Steiner Tree-Based Verification Approach for Handling Topology Changes in Self-Organizing Networks. In *International Conference on Network and Service Management (CNSM 2016)*, Montreal, Canada, October 2016.

[TATSC16c] Tsvetko Tsvetkov, Janne Ali-Tolppa, Henning Sanneck, and Georg Carle. Verification of Configuration Management Changes in Self-Organizing Networks. *IEEE Transactions on Network and Service Management (TNSM)*, 13(4):885–898, December 2016. DOI: 10.1109/TNSM.2016.2589459.

[TFSC15] Tsvetko Tsvetkov, Christoph Frenzel, Henning Sanneck, and Georg Carle. A Constraint Optimization-Based Resolution of Verification Collisions in

Self-Organizing Networks. In *IEEE Global Communications Conference (GlobeCom 2015)*, San Diego, CA, USA, December 2015.

[TNSC14a] Tsvetko Tsvetkov, Szabolcs Nováczki, Henning Sanneck, and Georg Carle. A Configuration Management Assessment Method for SON Verification. In *11th International Symposium on Wireless Communications Systems (ISWCS 2014)*, Barcelona, Spain, August 2014.

[TNSC14b] Tsvetko Tsvetkov, Szabolcs Nováczki, Henning Sanneck, and Georg Carle. A Post-Action Verification Approach for Automatic Configuration Parameter Changes in Self-Organizing Networks. In *6th International Conference on Mobile Networks and Management (MONAMI 2014)*, Würzburg, Germany, September 2014.

[TSC14] Tsvetko Tsvetkov, Henning Sanneck, and Georg Carle. An Experimental System for SON Verification. In *11th International Symposium on Wireless Communications Systems (ISWCS 2014)*, Barcelona, Spain, August 2014. Demonstration session.

[TSC15] Tsvetko Tsvetkov, Henning Sanneck, and Georg Carle. A Graph Coloring Approach for Scheduling Undo Actions in Self-Organizing Networks. In *IFIP/IEEE International Symposium on Integrated Network Management (IM 2015)*, Ottawa, Canada, May 2015.

[vdHEZ+16] Jappe van der Hel, Philipp Eichhorn, Reinier Zwitserloot, Robbert Jan Grootjans, and Sander Koning. Project Lombok. `https://projectlombok.org/`, 2016. Visited: September 2016.

[Wei15] Eric W. Weisstein. Minimum Vertex Coloring From MathWorld - A Wolfram Web Resource. `http://mathworld.wolfram.com/MinimumVertexColoring.html`, 2015. Visited: February 2015.

[YKK12] Toshiaki Yamamoto, Toshihiko Komine, and Satoshi Konishi. Mobility Load Balancing Scheme based on Cell Reselection. In *International Conference on Wireless and Mobile Communications (ICWMC 2012)*, Venice, Italy, June 2012.

[YL11] Dmitry S. Yershov and Steven M. LaValle. Simplicial Dijkstra and A* Algorithms for Optimal Feedback Planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3862–3867, September 2011.

ISBN 978-3-937201-56-6
DOI 10.2313/NET-2017-07-1
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)