Self-Healing in Self-Organizing Networks

Oliver Scheit
Betreuer: Tsvetko Tsvetkov
Seminar Innovative Internettechnologien und Mobilkommunikation SS2014
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: oliver.scheit@tum.de

ABSTRACT
Self-Organizing Networks are a way to manage the increasing number and complexity of current mobile networks. They provide functions to automatically configure network elements allowing for faster installation of new network elements. The domains SON also include the ability to self-optimize the network parameters and to detect and solve problems. The detection and diagnosis of degraded network elements is based on performance indicators provided by the individual network element. The diagnosis furthermore requires the combination of the knowledge of the correlation between the performance indicators and the corresponding faults. This paper shows an example framework that implements the detection and diagnosis of faults and provides an example of a SON-function that can find conflicts during the self-automated processes.

1. INTRODUCTION
Due to increasing complexity seen in today's networks, methods to efficiently manage them have to be introduced. One way to do so is to automate various parts of the system and create a so called "SON", short for self-organizing network. The goal of a SON is to configure, maintain and self-heal parts of the system based on information the system gathers itself, without needing manual assistance from the systems operator.

One usage area of SONs is mobile communication, in particular LTE (long term evolution), where a large amount of Base Stations are required for coverage purposes. Deploying a bigger number of network elements however means that there is a higher chance of failure in the systems network. This is the reason self-healing of self-organizing networks was introduced as a requirement for LTE by 3GPP, the 3rd Generation Partnership Project, to automatically identify faulty behavior.

This paper aims to give a short summary about the general idea and the domains of self-organizing networks and a more detailed overview about the self-healing aspect of SONs based on LTE network deployment. This includes the detection of a faulty element and the diagnosis of the root problem.

The document is structured as follows: in Section 2 self-organizing network domains will be explained and the benefits of automation will be shown. Section 3 will detail the general self-healing idea. Section 4 gives an insight into detection of degraded network elements and Section 5 will show different ways to diagnose the root-cause of problems in the network. Section 6 will show case how to implement an automatic self-healing framework. In Section 7 a SON-function to detect problems in the self-optimization process is outlined and Section 8 shows other possible uses for SONs. Section 9 will then conclude the paper.

2. SELF-ORGANIZING NETWORKS
Modern mobile communications networks tend towards end nodes with smaller range to be able to serve users with higher bandwidth and more stable connections, this drastically increases the amount of network elements to be deployed and maintained. Configuring, optimizing, monitoring and troubleshooting all individual cells therefore is an unpractical, if not impossible, task for the networks operator. To provide an efficient cost-effective network these tasks have to be at least partly automated so that the operator only has to supervise the parameter provided to him by the system and the SON processes. Such a self-managing network is commonly referred to as a self-organizing network. In mobile communications the domains of a SON include self-configuration, self-optimization and self-healing of the systems elements. [3]

2.1 Self-Configuration
To be able to efficiently install new hardware or software the deployment of network elements should be a *"plug & play"* system in which cells automatically download the necessary setup information and adapt to their environment. The SON self-configuration enables the system to add new elements to the existing network in real-time. The auto configuration sets up physical parameters like Physical Cell Identifier, transmission power and antenna tilt. In addition it is responsible for neighbor discovery and handover parameters. [3]

2.2 Self-Optimization
The goal of self-optimization is to find the most efficient parameters for a network at any given time. If the network environment changes over time, the system has to reconfigure parameters accordingly.[3] Some examples of self-optimization are

- Mobility Load Balancing (MLB) which enables cells to redirect part of the user generated traffic to neighboring cells. This allows the load to be evenly distributed across cells giving customers a better throughput and allowing an improved user experience. MLB also increases the overall capacity of the network and helps avoiding a congested cell while neighboring cells are unused.
- Energy saving allows cells to be deactivated during times of low traffic to reduce the power consumption of

doi: 10.2313/NET-2014-08-1_22

the network without impairing the overall network performance.

- Mobility Robustness Optimization or MRO. This function aims to minimize intra-LTE handover failures or unnecessary handovers to other radio access technologies. Typical scenarios include handovers initiated too early/late or to wrong cells.
- Inter Cell Interference Coordination enables cells to coordinate their signals to decrease interference i.e. cell A uses the lower part of the channel bandwidth while cell B uses the upper part

All of the functions above can be implemented as decentralized functions in each network element. [2]

3. SELF-HEALING

Self-healing is the least researched of the self-organizing functions as it is the most complex of the SON domains. Nevertheless it is still an important part of self-management as even though the network won't be able to recover a physical damaged cell, it can help identify which network element is not functioning as expected. The self-healing processes should contain:

- Alarm correlation, a diagnosis process triggered on alarms that tries to find the root cause of the alarm. If an possible cause is discovered, an automatic recovery action can be started to try to resolve the problem
- Sleeping cell or cell degradation detection, locate cells that don't transmit faults/alarms but still do not perform as planned
- Cell Outage Compensation; reconfigure neighbor cells temporarily to compensate the failure of another cell.

3.1 Self-healing use cases by 3GPP

The 3GPP, 3rd Generation Partnership Program has therefore defined sell-healing use cases.

Self-recovery of Network Element software: try loading an older software version or configuration to solve faults. This can be done multiple times until you reach a certain amount of tries. The process will then inform the IRP-Manager if the recovery was successful or not.

Self-healing of board faults: discovery of malfunctioning network element hardware. If there is a redundant hardware element the cell will try to activate the backup. If there is no redundancy and there is a loss of radio services, cell outage management will be started. The use cases for cell outage management include:

Cell outage Detection: The System detects a degraded, out-of-service or a "sleeping cell". A cell that does not serve user equipment with requested data even though users are connected to the cell and that does not raise any alarms to indicate the malfunction. To detect these kind of problems so called performance indicators are observed and irregularities are reported to the operator.

Cell outage Recovery: Based on the diagnosis an available recovery action is performed to restore the systems normal capabilities and the results will be reported.

Cell Outage Compensation: as stated above, the network reconfigures itself to compensate for potential coverage holes created by the failure.

Return from Cell Outage Compensation: After the fault has been resolved the cells that took part in the compensation process have to return to their pre-fault state to ensure the optimal configuration state.

3.2 3GPP self-healing process

The proposed general self-healing process is structured as follows: An input monitoring function continuously checks whether a performance indicator has violated its corresponding threshold. In case one or more did the associated self-healing process will be triggered. The process will then gather additional system information like performance indicators, system variables, test results and radio measurements. See chapter 4 for a more detailed explanation about the detection process. With this aggregated data an analysis function is started to detect the root cause of the problem. Depending on the result of the diagnosis, i.e. was a there a problem at all, corrective action is taken if the problem can be solved through a SON-function. It is advisable to back up the system configuration prior to executing the SON process however. Afterwards an evaluation of the new system state is initiated to see whether the problem has been resolved or not or even if the new configuration created new problems. These steps are repeated until the fault is solved or stop conditions i.e. a reset counter is fulfilled. The result of the self-healing process will be monitored by the Self-healing and Monitoring function that has access to data of all of the above mentioned process and allows the operator to control the execution of the self-healing process. [3]

3.3 Cell degradation management

Cell Degradation management is one of the most important parts of the self-healing process. Detection and Diagnosis of faults can reduce the costs for the operator and help improving the user experience.
The typical problems found in cellular networks are hardware or software faults, planning and configuration faults and environmental changes. Without SON functionality the degradation detection is only partially automated. The system will check itself for alarms, once an alarm is found the most common attempt to fix the cell is to reset it. If the problem is not resolved after a given number of resets the system will alert the operations department. The element will then be investigated remotely, in case this does not help solve the problem a technician will have to inspect the cell on-site and possibly replace hardware elements or the whole cell. Not all cell degradations cause alarms however, to see if a cell is working in an efficient, optimal way the optimization department will have to assess additional parameters provided by the cell. These are called KPI or PI, (key) performance indicators. To find malfunctioning cells thresholds for these parameters are set that should not by violated i.e. percentage of dropped calls. Base stations that breach the allowed limit or that show the biggest change in performance from one time period to another will then be selected for further investigation. [3]

3.4 Cell outage compensation (COC)

Cell outage compensation is one of the most crucial parts of the self-healing use cases. After the detection and diagnosis of a faulty network element neighbor cells will be configured to partly provide coverage for the area of the broken cell. The COC will be triggered when it is clear that the loss-of-service will persist for a prolonged period of time. To be able to do this several parameters of the neighboring nodes can be tweaked like antenna tilt, transmission power, beam forming and shaping. Since it is desirable to recover the service in the area as fast as possible to minimize user complaints the COC process will not try to find optimal parameters and settle for providing coverage in the affected area as fast as possible. After the COC process is finished additional self-optimization algorithms can be executed to find optimal settings for the new network environment. [3]

4. DETECTION OF DEGRADATIONS

The detection of faulty hardware or performance degradations plays an important role in the self-healing domain. Being able to reliantly detect problems in cells allows for faster diagnosis and reduces human resources. The detection algorithm does not equal the diagnosis process. Detection focuses on finding anomalous behavior and triggers an alarm if it found an unusual system state. The diagnosis process then gathers the information provided by the detection and concludes whether this is an actual fault or normal system behavior. The detection of degradation is based on event counters, KPIs and alarms. The process will determine if the provided performance indicators are in an acceptable range i.e. “healthy”. As these performance indicators are usually stochastically distributed statistical methods to define a healthy state have to be found. To do so you can create profiles for each of the KPIs that describe a range of normal, expected behavior. A KPI that leaves this range could point to degradation in the network performance. This approach has the advantage that the operator can define states that are normal and does not have to define states that are degraded, as these are often hard to define and statistically seen rather rare in overall network lifetime. The profiles can be split into three basic categories:

- Absolute threshold: this profile defines a threshold that should not be violated. Therefore the KPI should not exceed or fall short of. Examples for values that should not be exceeded are fault or failure rates like call drop ratio. An example for a rate that should not fall below a certain threshold would be the handover success ratio. See Figure 1
- Statistical profile: This profile defines an acceptable upper, maximum threshold that should not be exceeded and a lower, minimum threshold. These profiles can define a mean and it corresponding standard deviation that should not be violated. This profile can be symmetric or asymmetric. See Figure 2
- Time dependent profile: Much like the statistical profile there is an upper and a lower threshold that the KPI should comply with. The difference is that these thresholds fluctuate with time, meaning that acceptable borders will change during the time period. These profiles are typically user driven, e.g. traffic generated in the system. See Figure 3

While most of the task mentioned above, are already done automatically, the process to define what is faulty behavior and what is normal, but until now not encountered behavior has to be further refined to improve the accuracy of the fault detection. There are two different approaches for the anomaly detection. The first is the univariate, here a single performance indicator is observed and its specific distribution is considered to decide if it is in an acceptable range of its past mean and deviation.

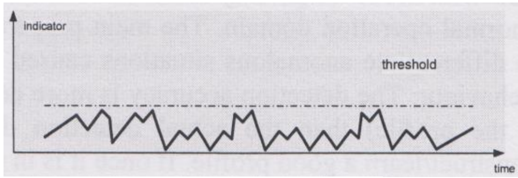

Figure 1: absolute threshold [3]

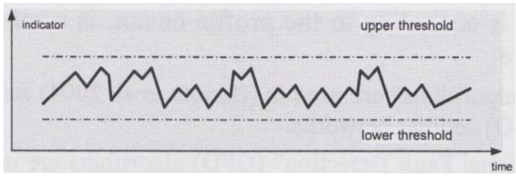

Figure 2: statistical profile [3]

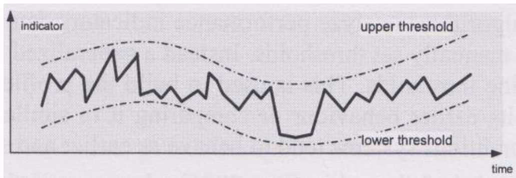

Figure 3: time dependent profile [3]

A drawback of this approach is that unusual behavior can be, depending on the profile, declared as a faulty behavior even though the cell is working fine. The other possible method is the multivariate anomaly detection where, as the name suggests, multiple performance indicators are considered and compared against each other to determine if the cell is working correctly. In this approach the performance of the cell as a whole rather than single KPIs can be evaluated, which makes it more reliable at avoiding false alarms.

The quality of a detection algorithm can be classified by several grades, as shown in [3]:

- The first is the detection accuracy, which gives an indication about the reliability of the detection algorithms. The goal of the detection process should be to have a high level of true negatives, i.e. no fault no alarm and true positives, i.e. a fault has occurred and an alarm was triggered and to avoid false negatives or false positive, i.e. an alarm was trigger even although there was no fault or worse there was a degradation but it was not detected by the process.
- Detection delay is the amount of time passed between a fault appeared and the detection of it. The faster the detection works, the faster the problem can be resolved or at least compensated.
- Other relevant metrics for operators may include severity indication accuracy that gives the operator an

doi: 10.2313/NET-2014-08-1_22

estimation of the overall severity of the degradation so that he may schedule maintenance accordingly. The detection process should give a reliable estimate of the severity, because simpler less performance impairing faults do not have to be resolved immediately, whereas grave faults like cell outage might need fast corrective action. Signaling Overhead describes the network load produced by the detection process between the individual cells, the cell and the network management and the cell and user devices, e.g. User equipment measurements.

- The last metric is the Processing Overhead, i.e. the amount of processing time needed for the detection algorithm. This typically depends on the amount of input data and can be exponential which poses a problem for network elements small processing capabilities.

5. CELL-DEGRADATION DIAGNOSIS

While the detection of problems in the network is at least partially automated nowadays the diagnosis of the corresponding root cause is still a manual task carried out by troubleshooting experts with years of experience. The general troubleshooting process starts with a notification of the expert by the detection system. The experts will then gather additional performance indicators and identify possible root causes for the problem. In case the problem can't be identified by these means a drive test will have to be performed on-site. An increase in network elements however makes manual troubleshooting a time intensive and cost expensive task for operators.

The complete automation of this process is the key goal of self-organizing networks. A self-healing system would require less maintenance and less troubleshooting personnel to manage bigger systems than before.

The challenge is to build a system that, like a human, can make decisions based on prior knowledge and can use its observations to include new similar cases to its database. To do so the system will have to learn a few basic symptom – root cause relations from existing human knowledge, this phase is called the learning phase of the diagnosis system. The next phase would be the assistive phase, where the diagnosis process would inform the troubleshooting expert about the most probable cause of the problem, but the execution of recovery action would still be done by the human operator. The root cause diagnosis will then be fine-tuned by knowledge of the troubleshooting expert. The last phase would be a completely autonomous system where the human operator only supervises decisions done by the self-healing process.

The critical component of this system is mapping symptoms to their respective root cause in an efficient, but reliant way. There are several ways to implement this association; some examples will be explained in short in the following sections, a more detailed view on these can be found in [3].

5.1 Rule Based Systems

The easiest way to implement this diagnosis is the rule based system. Every set of symptoms A is mapped to the root case B in a simple *"if (A) then (B)"* style of rules. Obviously a rule based system is easy to train as new fault cases can added as a new rule set. This system is very similar to human reasoning and easy to understand. It also proved to work efficiently in smaller deterministic environments, but the huge number of different parameters in mobile networks and their linked nature make these systems hard to fine-tune to effectively find problems in mobile networks. Additionally symptoms cannot always be associated to a single root cause, but with a probability. Rule based system only allow one specific case for a set of input parameters.

5.2 Bayesian Networks

Based on Bayes' theorem this system is able to include the uncertainty included in the performance indicators. Every root cause can be mapped to different symptoms and every symptom can be linked to different root cases. These links however do not always appear with the same frequency, thus they can be expressed with probabilities. The whole Bayesian network can be visualized by an acyclic directed graph to give a better overview of the individual dependencies for the operator. Since the direction of the tree does not matter operators can build the graph from the top-down. This means he can define new root-causes and add the accompanying symptoms, turning this into an easy to maintain and improve diagnosis system. The problem with this model however is that for each parent node several probabilities for its child nodes have to be defined. These probabilities are not always obvious or can be determined with the required knowledge for the system to work with the desired accuracy. Furthermore with a growing number of nodes the size of the conditional probability table grows exponentially in its size making it inefficient to compute and maintain. The Bayesian model can be simplified by two different models one is the naïve model, where only one root cause can be present each time. This simplifies the model because symptoms cannot be associated with multiple root causes which in turn decreases the amount of processing power required. The noise-or model allows multiple root-causes. It reduces the amount of space needed for the conditional probability table from 2^n to n by representing every root cause as a binary node that either exists or not. Than the system will measure the individual contribution of the root cause to each symptom.

5.3 Case Based Reasoning

Bayesian Networks heavily rely on the distribution of faults to determine the root-cause, but during normal operation of a network element these fault cases should be rather rare. This makes finding a correct probability for the conditional probability table nearly impossible as reliable data on faults cannot be provided or the statistical sample is so small that even minor deviations can trigger false alarms. Case Based Reasoning therefore tries to classify normal operation and will only react if there is a deviation from this "healthy" state of the cell. The basic idea behind a CBR system is to continuously build on the fault states already discovered by the system. The system starts with no knowledge of fault cases and monitors the performance indicators for anomalies, if an anomaly is found it tries to solve the problem by looking at previously encountered fault states with identical KPI levels and tries to solve the problem accordingly to previous solutions. If the root case was found and the problem was solved the system adds this new case with the associated parameters to its fault database. In case the problem was not resolved the system informs an operator to troubleshoot the cell. The result of this manual troubleshooting process will then be added to the existing cases creating an ever growing database of root cases with their respective symptoms.

6. AUTOMATIC DETECTION AND DIAGNOSIS FRAMEWORK

This Section will give a closer look into the working of an automatic detection and diagnosis framework proposed by Nováczki et al[4] and the exact mechanics behind the system. The system should be able to integrate into existing network environments without having to perform major modifications in the existing network. Additionally it has to be able to adapt itself to the existing system and its performance indicators and other available parameters. It is similar to case based reasoning, with the difference that it does not try to resolve issues by comparing it directly to older cases, but by assigning KPI footprints to so called targets and comparing these reports with the current measurements gathered by the system.

6.1 Detection

Instead of defining thresholds for variables that generate a binary result for the diagnosis algorithm, i.e. fault yes or no, a way to transform the KPI value into a continuous number in the range from 0 to 1 is proposed. Where 0 means perfectly healthy behavior and the 1 is asymptotically approached as the KPI state decreases. To build a profile of the KPI n consecutive samples x_i of the KPI are averaged to build one sample a_i. The KPI variable corresponding to the performance indicator K is called K^*. The profile will then be built using the mean and variance of the K^*. Assuming the x_i are independent and identically distributed the variable K^* should be normally distributed with

$K^*_{n\to\infty} \sim \mathcal{N}(\mu(K^*), \sigma^2(K^*))$

As a general example n should be bigger than 40 and more than k > 20 samples should be used, if the samples are not correlated. If the samples are correlated more samples are needed. After the profiling process has been completed, the current state of the KPI called $X^{(K)}$ is computed using the average of the latest n samples of the variable K^*. This current mean $X^{(K)}$ can then be transformed into a standard normal variable

$Z^{(X)} = (X^{(K)} - \mu(K^*)/\sigma(K^*))$

Where μ is the mean of the variable, this means the expected value and σ is the root of the variance the variable shows. The variance is the expected deviation from the mean.

After creating the profile K^* and current state $X^{(K)}$ of the KPI, a level function to evaluate the state and transform it into a number in the range from 0 to 1 has to be defined. The level of the KPI will be denoted by φ(K). The evaluation will be based on the desired thresholds of the KPI profile, i.e. upper threshold, lower threshold or both and the number of standard deviations -C allowed to result in the KPI level of 0.5. An acceptable upper threshold will be $\varphi_+(K)$ meaning that the KPI can decrease but should not increase. $\varphi_-(K)$ is the opposite where an increase in KPI levels is acceptable. Finally $\varphi_\pm(K)$ where neither an increase nor a decrease of the KPI level is desirable. The resulting functions are:

$\varphi_+(K) = \Phi\,(C + Z^{(K)})$

$\varphi_-(K) = \Phi\,(C - Z^{(K)})$

$\varphi_\pm(K) = \varphi_+(K) + \varphi_-(K)$

Where Φ is the cumulative distribution function of $\mathcal{N}(0,1)$, i.e. the standard normal distribution. The advantage of this approach is that the function does not check for a single violation of a threshold, but for a constant continuous decrease in the overall KPI performance. It also provides a uniform level function for all KPIs without the operator setting specific thresholds for single KPIs. The problem with this definition of a profile however is that most performance indicators are user behavior dependent, meaning that the profile has to adapt to different user behavior in different time periods of the day. Using a profile with a large standard deviation proves impractical as it impairs the detection process, therefore constructing different profiles for different time slots should be created. Too get a mean of a specific time t you can interpolate the means of the times t_1 and t_2 using a standard linear interpolation function. [4]

6.2 Diagnosis

Now that the detection of KPI levels has been defined the actual mapping of the levels to Targets has to be created. Targets can be corrective actions or root cases. This mapping is called a report. Each time a fault with the same symptom appears an identical report will be created and added to a database so that the frequency of a target can be measured as well. These reports can either be created by an expert for the system or added based on the solution of previous faults occurring in the network. Each report contains the subset of KPIs that were observed and the target that was analyzed by the specific report. There is a special target T_0 which means that the system is working perfectly healthy and its report R_0 that contains an empty set of KPI and the target T_0. No other report however is allowed to have an empty set of KPIs.

In order to find targets the system continuously monitors the KPIs and the "winning" target is the one that matches the most KPI states based on reports previously acquired by the system. Because not all KPIs are relevant to a target and not all reports for a target have the same KPI subset a likelihood function is introduced to evaluate which performance indicators are actually relevant to the target. This likelihood function is denoted as $l^T(K)$ and shows the relevance of the KPI. It is calculated by dividing the number of reports for target T that contain the KPI K through the overall number of reports for T. A likelihood of 0.5 means that the observed KPI was present in half of the reports for T rendering it useless as an indicator for T, because it does not appear to be linked to the target.

This can be further refined by balancing the KPI with a consistency function cf(x) = 2 * |x – 0.5|. This consistency function gives a weighted relevance of the KPI to a target. If the KPI K likelihood is close to 0 or to 1 for a Target T_i the KPI will have a high relevance, since you can say that whenever K is absent or present the target T_i is present too. To be able to compare different targets you have to quantify a score that is dependent on the KPIs. The score of a single KPI s(K) towards a the Target can be realized by multiplying the consistency function with a hamming distance:

$d^T(K) = \varphi(K)$ if $l^T(K) \geq 0.5$, else $1 - \varphi(K)$

The overall score of a target is the score of all KPI scores for T:

$S(T) = \sum s(K_i)$.

In case two targets T_1 and T_2 end up with the same overall score, the target with the higher overall number of reports will get the priority.

Due to the consistency contradictory reports should be checked as they can cause the KPI scores to become close to zero giving the

doi: 10.2313/NET-2014-08-1_22

target a lower score in the future. Contradictory results may also be an indication that there is an unknown error that should be investigated and added to the database. [4]

6.3 Integration into the Network

In general not all KPIs will be kept live to keep the amount of data an element creates to a minimum level, therefore not all KPIs should contribute to the initial detection of a problem. These inactive KPIs however could contain relevant data to the solution and need to be considered during the diagnosis. To resolve the problem an update of these KPIs is needed and should be initiated by the system. This is called an active measurement; they can be useful if a conclusion between different targets cannot be reached by the initial diagnosis process. In case the active measurement can't be initiated by the system itself it should notify a troubleshooting expert with an alarm that an active measurement is required to resolve a problem. To find the most suitable KPI that will most likely resolve the stalemate the KPI with the maximum likelihood for one of the two cells should be considered, as it will most likely be the most impacting on the overall score of the respective targets.

One advantage of the proposed framework is that it can, identical to case base reasoning, start with an empty knowledge base on targets and add targets and their corresponding reports during its lifetime. This means that the operator only has to initially do the troubleshooting process and can let the system take over if it has enough targets and reports to guarantee optimal operation. The management of the expert knowledge generated by the system could be stored on a management server and set to synchronize regularly with the cell to keep the information up-to-date. It would also enable the troubleshooting expert to edit the reports and add a root cause to help the system. [4]

6.4 Possible improvements to the system

To further reduce the amount of human interaction required by the system an extension to this system can be made where the operator only has to specify time slots where normal operation was detected. The framework can then implement a so called two-sample Kolmogorov – Smirnov test, that can compare two random distributions to check whether they have the same distribution or not.

Therefore it compares the empirical cumulative distribution functions, ECDF, and computes the maximum distance between these two functions. See Figure 4. [5]

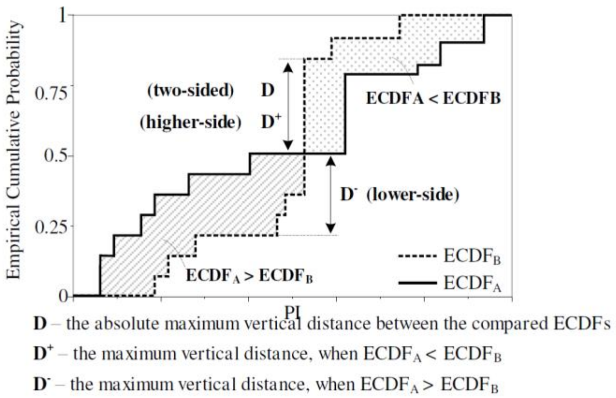

Figure 4: ECDF – comparison [5]. D is $\max\{D^+;D^-\}$, D^- is the max distance of $ECDF_A > ECDF_B$. D^+ the max of $ECDF_B > ECDF_A$

The ECDF is the cumulative addition of the probabilities over all events. Therefore the ECDF ranges from 0 to 1.

The distribution of a KPI profile is its corresponding ECDF function, which means that an ECDF will have to be computed for every KPI that will be considered in future diagnosis processes. The operator now only has to mark the time slots in which the KPIs and the overall system operation were considered "healthy". The input is then divided into fragments. For each of these fragments the ECDF is computed and saved as a possible profile candidate. To reduce the number of profiles only the most representatives are stored however. Then a process starts to find the minimum number of profiles that cover all the profile candidates. If a profile covers a candidate is decided as presented earlier by the two-sample Kolmogorov-Smirnov test. [5] To detect if one PI shows unusual behavior its latest N samples from the observation window are taken and the Actual-ECDF of the PI is computed. The detector then computes the maximum distance of this A-ECDF from all the profiles in the database of this particular PI. The minimum distance will be considered as the best matching example and it will be compared to a similarity threshold. Distances below this threshold can be defined as "healthy" operation, since fluctuation in parameters is normal to a certain level. Since there are different thresholds for each PI different methods to compute the distance will be used.

- Success Indicators: if the values are higher than expected there is no fault present to therefore the distance of the lower side will be use D^- (Figure 4)
- Failure Indicators: Values should not fall below a certain threshold. The relevant distance is the high sided D^+
- Neutral: Values should be between a certain range which means that both distances will be considered in diagnosis

An unknown distribution can either mean normal operation that differs from all profiles gathered before or the system is in a faulty state. This has to be decided by the operator that marks this new distribution as a new error or normal behavior. [5]

7. SON-FUNCTION COORDINATION

Several SON-functions might interfere with each other at run-time, e.g. the SON MRO (mobility robustness optimization) and the CCO (coverage and capacity optimization) where the MRO might try to resolve handover failures, but these failures are caused by a coverage hole, for which the CCO function would be responsible. To prevent this hierarchies of priorities could be introduced for SON-functions, but this could lead to deadlocks where a high priority function which is unable to fulfil its goal blocks a low priority function that could have resolved the problem. Another approach however is to implement another SON-function that monitors the execution of the other functions and tries to find locks in which no improvement in the network is seen, even though parameters are constantly adjusted. This function can be called a SONOT (SON Operational Troubleshooting) that has two main aims, to monitor the execution of other functions and find unresolvable problems and to help resolve the problem automatically.

To detect a problem with a SON-function different approaches can be used a state-based approach that only considers the current system state and a history based approach that allows considering

the influence of changes to parameters to the overall system performance. It also enables the system to predict whether a function will be able to fulfil its target or not. It can also track if a function continuously tweaks parameters, an indicator that it is not able to achieve its desired state.
The SONOT function will then try to analyze the functions involved and the root cause of the problem. It will also try to block all non-relevant SON-function execution to block any interference into the self-healing process. It will then call an more suitable SON-function to resolve the problem or if the problem is not resolvable by the system it will notify the network operator that the problem cannot be solved manually and requires manual troubleshooting. [1]

8. Other Self-Organizing Systems

Of course the applicability of SONs is not limited to mobile communications, another example of this network type is the freifunk network that is currently in development in Germany and Austria. This project tries to create an open network that provides internet access to all participants. To do so the nodes have to be able to find a path to an internet distributing node and deal with constant changes to the network topology in a self-organizing manner [6].

A different application from wireless networks would be road networks. As shown in [7] by Prothmann et al. local, distributed routing decisions can be applied to improve the traffic flow in cities in the future. The paper shows that the routing algorithms used in the internet like Link State and Distance Vector can be modified to use them to control traffic. The traffic light at intersections equal routers and can be used to control how many vehicles can pass through a given route and optimize time to a cities important locations.

9. Conclusion

Self-Organizing Networks are an efficient way to manage the increasing complexity in cellular networks seen today. They offer an easy way to install new cells for the operator with their plug & play style self-configuration and reduce the amount of maintenance required through their self-optimization capabilities. At the same time they possibly improve the user experience beyond a level possible by manual configuration of parameters. As described in this paper there are also ways for the system to automatically deal with problems occurring during the run-time of the system. This reduces the amount of time needed to detect problems in the network and can decrease the amount of necessary drive tests in the network. It can also help detecting problems more reliably than it would be possible with manual troubleshooting. This paper also showed the requirements needed like reliable performance indicators and the corresponding profiles. The different diagnosis approaches are introduced and their respective weaknesses and advantages are explained. Then a conceptual framework is showcased and the way it handles the detection of degradation and how it build its profiles is explained. The diagnosis process and its advantages against other similar approaches are shown. In the last Section an approach to fault handling of SON-functions is outlined that can detect problems in the interaction of SON-functions or find functions that cannot reach their desired objectives. Other applications of Self-Organizing Systems are then given as a conclusion to the paper.

10. Literature

[1] 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), Sydney, Australia, June 2014.

[2]Chris Johnson. Long Term Evolution IN BULLETS. Edition 2, version 1. ISBN 9781478166177. Amazon Distribution GmBH, Leipzig, 2012

[3]Seppo Hämäläinen, Henning Sanneck, Cinzia Sartori. LTE Self-Organising Networks (SON): Network Management Automation for Operational Efficiency. ISBN 9781119970675. John Wiley & Sons, 2012.

[4]Péter Szilágyi, Szabolcs Nováczki. An Automatic Detection and Diagnosis Framework for Mobile Communication Systems .IEEE transactions on network and service management, VOL. 9, NO. 2, JUNE 2012.

[5]Szabolcs Nováczki, S., "An improved anomaly detection and diagnosis framework for mobile network operators," Design of Reliable Communication Networks (DRCN), 2013 9th International Conference on the, vol., no., pp.234,241, 4-7 March 2013

[6] http://freifunk.net/, accessed 29.07.2014

[7] Holger Prothmann, Sven Tomforde, Johannes Lyda, Jürgen Branke, Jörg Hähner, Christian Müller-Schloer, Hartmut Schmeck. Self-Organised Routing for Road Networks. Self-Organizing Systems. 6th IFIP TC 6 InternationalWorkshop, IWSOS 2012 Delft, The Netherlands, March 15-16, 2012 Proceedings. ISBN 978-3-642-28582-0

doi: 10.2313/NET-2014-08-1_22