Automated Troubleshooting of a UMTS-WLAN Test Platform

Ronan Skehill
University of Limerick
ECE Department
Limerick, Ireland
ronan.skehill@ul.ie

Sean Mc Grath
University of Limerick
ECE Department
Limerick, Ireland
sean.mcgrath@ul.ie

Carlos Antonio de Ramos
University of Limerick
ECE Department
Limerick, Ireland
wireless@ul.ie

ABSTRACT

Amongst troubleshooting tasks, correct diagnosis of faults is a complex and time consuming task. Bayesian Networks provide a modelling approach suitable to cater for the uncertainty inherent in human reasoning. The method presented in this paper, valid towards any system, is applied to a UMTS/WLAN test platform which replicates a real world cellular networks. The results conclusively show Bayesian Networks automatically diagnoses faults in the test platform with a high success rate.

Categories and Subject Descriptors

H.1.1 [**Systems and Information Theory**]: Value of information; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Measurment, Analysis.

Keywords

Automated Troubleshooting, Test Platform

1. INTRODUCTION

Troubleshooting can be defined as a process where the source of the problem is isolated and then fixed. This is typically through a process of elimination whereby possible sources of the problem are investigated and eliminated beginning with the most obvious or easiest problem to fix. Troubleshooting has been used in many different disciplines including; medicine [1], forensic science [2], printer customer care and cellular networks. Automating and improving this task is an emerging topic. Automation is particularly useful in telecommunication field, where large networks are controlled and monitored to ensure operation. Current cellular telecommunication networks now consist of several radio networks integrated together to provide a complex system in terms of architecture and management. The large number of possible faults; software, hardware, configuration and environmental, makes troubleshooting a complex task. The automation of troubleshooting in general, and fault diagnosis in particular will allow mobile operators to alleviate the burden of troubleshooting teams and to shorten the time necessary to identify faults and thus reduce the time in which the network suffers from poor performance. The need for troubleshooting is paramount as it ensures the network is managed and operating correctly. As with other disciplines current cellular network management [3] troubleshooting is a manual process and executed by experts in the RAN part of the network. These experts are personnel dedicated to carry out daily a series of checks in order to identify the faults. This troubleshooting process usually uses the principle of eliminating likely problem causes in order to single out the actual one. To do this, the troubleshooter needs to check several applications and databases and also needs to analyse performance indicators, cell configurations and alarms. The more experience the troubleshooter has results in a faster identification of the fault. Furthermore, the type of information available and the quality of the tools displaying relevant pieces of information can determine the speed of the work. This means that, in addition to a good understanding of the possible causes of the problems, a very good understanding of the tools available to access the sources of information is also required. It is difficult to transfer and share an expert's knowledge, an automated method for diagnosis that could learn over a period of time would alleviate the risk of losing expert knowledge.

1.1 Motivation

There are several reasons for developing a troubleshooting tool for systems, the main ones are; fast accurate diagnosis and retention of knowledge. Limited fidelity and flexibility of simulators has prompted researchers to build wireless network test platforms for realistic testing [4]. Test platforms have been developed as part of research and academic projects, examples include Roofnet, ORBIT, WHYNET etc., to provide a means of testing and evaluating new algorithms. The platforms developed are often complex as they replicate current networks to a fine level of detail. Generally, small teams develop test platforms and know every aspect and would be considered as 'experts'. The nature of research centres and universities see a high turnover of researchers and experts taking with them the expert knowledge of the platform. This can leave platforms redundant, losing hun-

TRIDENTCOM 2008, 17th – 20th Mar 2008, Innsbruck, Austria.
 ISBN 978-963-9799-24-0
DOI 10.4108/icst.tridentcom.2008.2989

dreds of person months of work. Further motivation is the fact test platforms require several hundred configuration files which users of the test platform have access to. The user can change certain files/parameters in order to evaluate a new algorithm, protocol etc. The problem arises when a user accidently sets a file/parameter to an incorrect value and then runs evaluation tests. The user has no way of knowing that the test platform is not running correctly with the guidance of expert users. A tool that could retrain the experts knowledge and diagnosis a fault automatically, only looking at the output of the test platform, would give the user confidence in the results obtained.

The advantage of using troubleshooting techniques on a test platform is multifold. Firstly, the test platform can be quickly diagnosed if a fault occurs. Secondly, the test platform provides a means of evaluating new troubleshooting techniques. Finally, since test platforms are based on current standards every effort has been done to replicate real network such as UMTS, WLAN etc., studying the effectiveness of troubleshooting techniques provides insight to how they will behave in a real network.

1.2 Paper Outline

The paper is structured as follows. Section 2 provides a guide to automated troubleshooting techniques i.e, Bayesian Networks, and describes the benefits of automating troubleshooting techniques. Section 3 describes a UMTS/WLAN test platform that is built according to 3GPP and IEEE standards. The platform consist of several interconnected modules that needs configuration and produces key performance indicators (KPI) similar to real world networks. Section 4 introduces common faults into the test platform and acquires KPI knowledge from normal and fault behaviour. This section also presents a step-by-step guide to the diagnosis model. The results from six faults introduced into the test platform and solved by the diagnosis model are presented in section 5. Finally in section 6 conclusion on the applicability and accuracy of Bayesian networks in automated troubleshooting is provided.

2. AUTOMATING TROUBLESHOOTING

2.1 Naïve Bayesian Networks

The concept of conditional probability can be used to solve a range of problems [5]. Consider a simple network management scenario with five discrete random variables the problem must handle 2^5- 1 discrete parameters. However in a real network situation a problem could have over 20 discrete random variables resulting in a joint distribution of over 2^{20} and this becomes unmanageable [3].

Consider the following parameters; Ci denotes a cause for bad functioning in the network, Sj as a symptom that can contribute to assessment of the network quality, and E is evidence from a set of N symptoms Sj. If, given certain values for the symptoms, the probability of each cause is calculated, the diagnosis consists of identifying the cause with the highest probability. This process can be seen as a classification process in which each class and attribute corresponds to a given cause and symptom, respectively. Using Bayes' rule [6], one can calculate the probability for the cause C_i to occur given the set of observed symptoms. Typically, certain conditions on the diagnosed network are known a prior, such as the type of services provided, the number of frequencies used by the base station, or any other a priory knowledge. The set of conditions is denoted by D. This can be described as:

$$P(C_i|E,D) = \frac{P(C_i|D)P(E|C_i)}{P(E)} \tag{1}$$

It is assumed in the formula that the symptoms are independent of the conditions. Calculating the joint likelihood distribution $P(E|Ci)$ is difficult and impractical. It is at this point that Naïve Bayesian Network can be used, which makes the assumption that symptoms given the causes are independent. With this assumption, it is only needed to specify the likelihood distribution of each symptom, $P(Sj|Ci)$, separately. This assumption is an approximation however the Naïve Bayesian classifier remains efficient even with significant dependencies between the symptoms. Hence, the classification in the diagnosis process will be performed using the approximation:

$$P(Ci|E,D) = \frac{P(Ci|D)\prod_{j=1}^{N} P(Sj|Ci)}{P(E)} \tag{2}$$

With this assumption, a Bayesian Network structure could be created to encode the assertions of conditional independence. The following example uses 2 to solve a simple problem with 2 faults and 3 KPIs. The evidence, symptoms and conditions are invented to highlight how the model works.

2.1.1 Example

Using KPIs [7], common to real networks and the test platform, an illustrated example of a system operating normal which has 'No Fault' (NF) and with two imaginary faults namely; a hardware failure (HW) and bad parameters value (BP). The aim of the example is to explain how Bayesian newtworks work by ranking these faults given some symptoms. Consider the following KPIs.

- **Block Call Rate (BCR)** This is the number of blocked calls during a measurement period divided by the total calls attempted during the measurement period.
- **Dropped Call Rate (DCR)** The number of calls that have been established and then disconnected.
- **Ping-Pong(PP)** The effect occurs when a mobile moves closer and further from a cell boundary causing frequent handovers.

Using the naïve Bayesian rule as in Equation 2, we can rank each fault given KPIs as:

$$P(Fault|BCR,DDR,PP) = \frac{P(Fault)P(BCR|Fault)P(DCR|Fault)P(PP|Fault)}{P(E)} \tag{3}$$

The same prior probability for each fault is established in this example. Then:

$$P(NF) = P(BP) = P(HW) = 1/3 \tag{4}$$

Before creating the Bayesian network the conditional probability of each KPI given each fault needs to be calculated. As KPIs are continuous values, it is impossible to establish

a probability an each infinite number of values that indicators can reach. So using a process called *discretization* each KPI is divided in two or more states according the numerical value.

Returning to the example, each KPI is *discretized* into three states, *NORMAL*, *HIGH* and *VERY HIGH*. It is assumed that hardware failure (HW) produces an abnormal BCR and PP values. If there is a bad parameter (BP) fault then DCR and PP are forced to abnormal states. There is no association between HF and BCR nor BP and DCR. In Table 1, all probability values and the different states are invented to illustrate and understand how the model works.

Table 1: ***The probabilities of KPIs BCR, DCR and PP given each fault***

BCR	**NF**	**HW**	
NORMAL	0.85	0.01	
HIGH	0.05	0.09	
VERY HIGH	0.1	0.9	
DCR	**NF**	**BP**	
NORMAL	0.7	0.05	
HIGH	0.2	0.15	
VERY HIGH	0.1	0.8	
PP	**NF**	**HW**	**BP**
NORMAL	0.8	0	0.1
HIGH	0.15	0.03	0.3
VERY HIGH	0.05	0.97	0.6

It is possible to calculate the posterior probability of each fault. As each of the three KPIs could be in any of these three states, there is 3^3=27 cases per fault. The whole Bayesian network is composed by 81 different posterior probabilities. For clarity, only one fault scenario is studied in this example. The scenario selected and KPIs values are shown in the Table 2:

Table 2: ***Fault Scenario: BCR in a 'Very High state'***

KPIs	**CURRENT VALUE**	**STATE**
BCR	0.949	VERY HIGH
DCR	0.014	VERY HIGH
PP	0.069	HIGH

Using Bayes formula the exact probability can be calculated as:

$$P(Fault|BCR_{vh}, DDR_{vh}, PP_h) = \frac{P(Fault)P(BCR|Fault)P(DCR|Fault)P(PP|Fault)}{P(E)} \quad (5)$$

The $P(E)$ term can be regarded as a normalising or scaling factor, and the $P(Fault)$ depends the number of faults and their probability distribution, in this example it's 1/3. Using data from Table 1.

- $P(BP)$=1/3
- $P(BCR_{vh}|BP) = 0.1$ It is 0.1 because the assumption is that if there is no association, the *default* probability of $P(BCR_{vh}|NoFault)$ is used
- $P(DCR_{vh}|BP) = 0.8$
- $P(PP_h/BP)$= 0.3
- $P(E)$ using the chain rule :

$$\begin{aligned} P(E) &= P(BP)P(BCR_{vh}/BP)P(DCR_{vh}/BP)P(PP_h/BP) \\ &+ P(HW)P(BCR_{vh}/HW)P(DCR_{vh}/HW)P(PP_h/HW) \\ &+ P(NF)P(BCR_{vh}/NF)P(DCR_{vh}/NF)P(PP_h/BP) \\ &= \mathbf{0.0094} \end{aligned} \quad (6)$$

This result and the state information is introduced into formula (5) and the following *fault* ranking is obtained.

- **1. Bad Parameter** $P(BP)$=0.851 This is the most likelihood cause.
- **2. Hardware Failure** $P(HW)$=0.095 This is the second most likelihood.
- **3. No fault case** $P(NF)$=0.053

The results conclusively show that a Bad Parameter is the most likely cause of the fault with 85%. The hardware fault is ranked 2^{nd} with a 9.5%. Finally, there is a 5.3% that the scenario is a no fault situation. The same method is used to determine faults in a UMTS-WLAN test platform, before illustrating the process a brief overview of the test platform is presented.

3. TEST PLATFORM

The UMTS-WLAN test platform [8] is a hardware/software platform that emulates an integrated UMTS-WLAN network system. This system includes multimedia terminals, UMTS and WLAN functional entities and IP connectivity. The test platform has the ability to support real-time multimedia calls in a large multi-user, multi-service UMTS system with WLAN cells embedded in the RAN and produces associated KPIs. The test platform spans nine PC's running the Linux operating system. From the UMTS domain there are test platform machines which replicate UE/MT (User Equipment/Mobile Terminal), UTRAN (Universal Terrestrial Radio Access Network), CN (Core Network) and Application Server functionalities. Similarly there are machines serving as entities from the WLAN domain in the form of an Access Point and WLAN client. At the centre of the test platform is a single, real, fully interactive client terminal, capable of utilising both UMTS and WLAN networks. Complete UMTS and WLAN protocol stacks have been constructed for this client terminal or 'reference user'. The test platform then emulates the effect that multiple users have on a real user's traffic. The test platform aims to reproduce the real behaviour of a single reference user terminal with more accuracy than a simulator but with less implementation complexity than a real system. Testbed experiments are scenario driven with scenarios being defined by user mobility patterns and various service profiles, it is therefore essential that the testbed is correctly configured and operating within its normal range.

The test platform is implemented as a set of higher layer, lower layer and RRM functions distributed across a number of PCs. Each machine runs a set of modules that replicate common UMTS/WLAN functionalities using a very specific programming philosophy i.e. mono-task approach, common procedures for communication with other modules and well defined interfaces. Each of these modules have configuration

files and settings which are set to reproduce 3GPP standards. The test platform is organised then as a set of modules tightly connected to a control structure known as the Communications Manager (CM). The CM manages all the test platform at module level, providing support for inter-module communication and a text based console for module control. The dissemination of these modules and the interfaces connecting them is shown in Figure 1. The test platform comprises five main blocks: the UE/MT, UTRAN, Access Point, CN and Application Server [8].

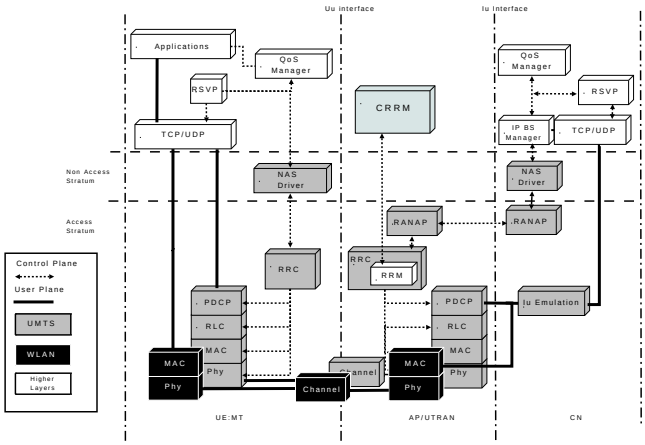

Figure 1: Testbed Modules

Each module in the test platform is associated with a set of control parameters that define how that module operates. These parameters can be adjusted pre-simulation using a management tool. One of the most important modules in the test platform is the RRM module found in the UTRAN. The RRM Module implements RRM algorithms like admission and congestion control, power control, handover management and transmission parameters management. The effect of the emulated UMTS users has been included inside the RRM Module. RRM strategies are executed inside the RRM module for all the users in the system as if all users were real like the reference user. The tunable parameters associated with the RRM module include the number of simulated users for each available service in the test platform (video calling, email, internet), the speed and mobility patterns of those users, activity factors, target BLER values for each service, load factors for uplink and downlink amongst other algorithm data.

The functions of RRM module ('701') as seen in situ by Figure 1 is shown in greater detail in Figure 2. The module is at the core of the test platform and is used in every evaluation. The module is complex as all RRM parameters regarding the single user under test and the RRM behaviour of the other virtual users. Currently the module has over 130 different parameters. Examples include propagation characteristic, number of users and type of user (realtime, streaming etc.) Each of these users can then be assigned a speed and a mobility pattern. RRM characteristics and targets can be set in the module also, for example the admission control strategy, congestion control and load factor targets. Properties regarding the radio can also be set in this module for all users. Examples include; NodeB power, mobile station power, the Block Error Rate (BLER) range, the signal to noise ratio ($\frac{E_b}{N_o}$), signal level for handover, Pilot levels etc. If parameters in this module are set incorrectly they can distort the results/KPIs obtained from the test platform.

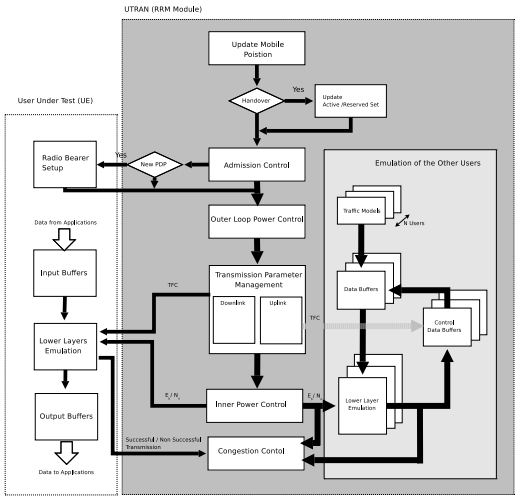

Figure 2: RRM Module 701 in the testbed

3.1 Scenario

As described in the previous section, several modules in the test platform require parameters to be configured/adjusted. In normal conditions these parameters are set to the default settings as outlined by 3GPP standards. In order to produce normal KPIs for the test platform a sample user scenario is used. A proportion between realtime (RT), streaming (STR) and background (WWW) users were selected to give a realistic representation of wireless users in a cellular/WLAN network.

- **Real time (RT):** 1500 users, all of them moving at $3Km/h$.
- **Streaming (STR):** 300 users, all of them moving at $3Km/h$.
- **WWW:** 300 users, all of them moving at $3Km/h$.
- **Best Effort:** 0 users, this parameter is not considered.

In all tests executed, faults are introduced in the RRM module. When it is necessary to create a fault and simulate the network, a parameter is artificially changed in this module. Initially the rest of the parameters of the test platform is set to default in order to obtain knowledge from the test platform.

4. KNOWLEDGE ACQUISITION

Gaining knowledge from any system follows a process as shown in Figure 3. Selecting the faults category is a significant problem as wireless test platforms and communication networks have a wide range of faults. Independent models for each fault can be developed. In this case test platform experts have picked and categorised several faults. Defining consists of selecting causes $(C_1, ..., C_K)$, symptoms $(S_i, ..., S_M)$, state of symptoms (s_i) and conditions (D_r^i). These are generally stored in a database. When defining the

cause-smyptom relationship, the cause is related to symptoms and the condition associated with the cause. For each continuous symptom, intervals or threshold must be determined. Finally, specifying the probabilities can be aided by expert knowledge.

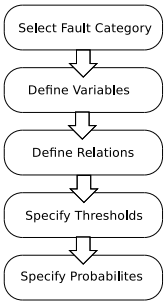

Figure 3: Knowledge Acquisition

In order to obtain the range of a KPI, the test platform is configured with all normal values for all modules. Simulations are executed and the test platform is allowed to initialise and run for an extended period of time. This process is repeated for several runs to obtain average values. Three runs determined statistical significant average values, however in some cases further runs were needed. The duration of the simulations is not a necessary parameter to create the model, but in order to achieve a converged parameter value from the test platform, a minimum duration of 15 minutes is used to determine the parameter. The time value however is used in calculating the admission request and rejects per second. Several simulations of each fault is needed until sufficient learning data is collected to create the model selecting appropriate thresholds.

4.1 Selecting Faults

In order to demonstrate the use of Bayesian networks, a wide range of faults are chosen from the the test platform. For example, if more than 10 faults are chosen the time needed to execute and obtain all the results from the test platform is increasing exponentially, however it will not give the Bayesian network any more information. Therefore it is necessary for experts to selectively pick faults and assigning them to categories. It is at here where the initial transfer of the expert knowledge to the Bayesian network is accomplished. In the case of this paper all faults belong to the RRM category as all the faults will be created in the RRM module. The selected potential parameters where faults can occur in the test platform RRM module are described briefly below and listed in detail in [9].

- **No Fault (NF)** In order to develop a troubleshooting model, several tests are executed in a normal state to determine normal KPI ranges.
- **BLER Target RT Uplink forced to an incorrect minimum value:** The Block error Rate is required for each simulation. Specifically, this parameter refers to the real time service. A minimum value affects the adapting power control.
- **BLER Target RT Uplink forced to an incorrect maximum value:** The same as previous fault except this is a maximum value.
- P_{ms} **Max forced to an incorrect value :** The Mobile Station maximum transmission power.
- P_{ms} **Min forced to an incorrect value:** The Mobile Stations minimum transmission power.
- P_{bs} **Max forced to an incorrect value:** Base station maximum transmitted power. This value is measured in dBm, and indicates the maximum power that could transmit the base station.
- **Pot Pilot forced to an incorrect value (PP min):** Base station Common Pilot Channel (CPICH power). In UMTS, CPICH is a down link channel, broadcast with constant power. This power usually is a percentage of the total Node-B power.

Table 3 represents normal values for the parameters in the test platform according to 3GPP specification.

Table 3: ***Normal parameter values***

FAULT	NORMAL VALUE
BLER RT MIN	0.01
BLER RT MAX	0.04
P_{bs} MIN	43 dBm
POT PILOT MIN	33 dBm
P_{ms} MAX	21 dBm
P_{ms} MIN	-44 dBm

4.2 Defining Variables

In the diagnosis model, 13 KPIs were selected to be used and then monitored in each test. Initially a further 10 were selected but then rejected as the first tests indicated no association with the fault, so they were not real KPIs, a similar selection process is described in [10].

The KPIs selected are:

- **Active users:** Is the average number of users using WLAN/UMTS network at any one instance.
- **Load factor Uplink (LoadF(u)):** The Uplink spectral efficiency of WCDMA cell is commonly captured by the uplink cell load factor. It is a function of the number of instantaneous users.
- **Load factor Downlink (LoadF(d)):** Is defined as a function of the number of simultaneous transmissions and their characteristics.
- **F value for Downlink admission (fdl):** The noise figure in dB for new admissions in the network.
- **Real Time BLER Uplink(RTBLER(u)):** BLER is Block error rate. The main difference with BER (Bit Error Rate) is that the first detects errors on transport block level and the BER only at a bit level.
- **WWW BLER Uplink:(WWWBLER(u)):** The same as previous but with reference to WWW traffic.
- **Real Time request per sec. (RTReq/s):** Number of total real time admission requests divided by the duration of the test.

- **Real Time rejects per sec. (RTRej/s):** Number of total real time admission rejects divided by the duration of the test.
- **WWW request per sec. (WWWReq/s):** Number of total WWW admission requests divided by the duration of the test.
- **WWW rejects per sec. (WWWRej/s):** Number of total WWW admission rejects divided by the duration of the test.
- **Streaming requests per sec. (STRReq/s):** Number of total streaming admission requests divided by the duration of the test.
- **Streaming rejects per sec. (STRRej/s):**Number of total streaming admission rejects divided by the duration of the test in seconds.

With test platform faults selected and appropriate KPIs identified the next process in automated troubleshooting is to learn the relations and normal operating ranges for KPIs. To acquire this data, the testbed is allowed to run in a normal state for extended period of time and KPIs are monitored. This data is often referred to as the models "Learning Data".

4.3 Learning Data and Defining KPI Relations

As illustrated by the Table 4 all KPIs have different values for each test which is typical for test platforms and real systems. For each fault, only parameters with relative difference of more than 15% from the average are shown. Mathematically it is expressed as:

$$\frac{ParamAvrg_{normal} - ParamAvrg_{fault}}{ParamAvrg_{normal}} 100 \geq 15 \qquad (7)$$

The first fault to acquire knowledge on is ***BLER Target RT Uplink*** forced to a minimum value. From Table 5 changing the value of modifies output BLER in the uplink. Furthermore, it affects the Load factor, the fdL and the admission rejects for real time services. Changing the ***BLER Target for realtime*** to be a maximum value the relationship between KPIs are different as shown in Table 6.

In the case of the fault P_{ms} ***Max forced to a minimum value***, almost all parameters can be considered KPIs. Only the values related to streaming do not change. For this special case more than three tests is needed to achieve confidence in the results. The results are illustrated in Table 7.

The next fault evaluated is P_{bs} ***Max forced to a minimum value***. In this case all KPIs are far from the normal case averages and it expresses a serious fault in the test platform. For example there is no Load factor, fdL, RT rejects, WWW request or rejects. In all the tests the BLER in real time for downlink is one and in the uplink approximately zero. The BLER in www services is zero. The rest of the KPIs values can be viewed in Table 8.

The next fault is produced by changing the values ***Pot Pilot*** to a minimum value. As shown in Table 9 in the main changes appear in the load factor downlink and in the fdl KPI. Furthermore, there are differences in the number of rejects in real time and www services.

The last fault is to force the P_{ms} ***Min to a maximum*** value. Significant differences in almost all the KPIs appear, Table 10 illustrates the results.

4.4 Defining Thresholds

Each KPI needs to be discretized in a range of states, as explained in the previous section. In this paper there are thirteen indicators and it is necessary to decide how many states are necessary and which is going to be the general rule to select it.

Looking at the faults and their dependence to the KPIs it is evident that three states is not sufficient. Firstly in some faults the value of the KPI is smaller than the no fault case, so a low state must be needed. Secondly in some KPIs depending on the fault five states can be clearly identified. For example the KPI fdl, the average in normal case is 1.41, whereas when the fault is P_{bs} *Max forced to a minimum value* is always zero, in some faults the order of magnitude is only some tenths, and when the fault is *Pot Pilot forced to minimum value,* the order of magnitude of ten. So in this KPI at least four states is needed to be created.

To improve the accuracy of the model some of the KPIs were divided into five states, others into four and the remainder into three states. The different states are classified as Very Low, Low, Normal, High and Very High. For KPIs with three states, Normal High and Very High are used, similarly for KPIs with 4 states, Low, Normal, High and Very High. Only the Very Low state is used for KPIs with five states.

Regarding the selected thresholds, it is difficult to find a rule that suits all KPIs. The thresholds were selected using the knowledge learned from normal, fault tests and input from the test platform expert. Table 11 lists each KPI and the threshold for each state.

4.5 Specifying Probabilities

A general way of specifying probabilities is translating verbal expressions to mathematical representation. For example, the verbal expressions "Almost Certain", "Likely", "Fifty-Fifty", "Improbable" and "Unlikely" can be translated to probabilities of $0.85, 0.7, 0.5, 0.3$ and 0.1. In the case of the test platform all probabilities of introducing a fault are equal, in this case each fault probability (including No Fault) are given a probability of 0.142 which gives a total probability of ≈ 1.

5. FAULT DETECTION

Illustrated in Table 12 are example configuration errors found by experts in the RRM module. These faults were reintroduced into the test platform by an independent user. The diagnosis team were not aware of the faults and only used the KPIs obtained to diagnosis the fault offline.

Several simulations using the test platform are done per each fault in order to achieve statistical significance. The simulations are done using the same scenario as in the learning data, i.e. 1500 users in real time services, 300 in WWW and Streaming services and 0 best effort users, all moving at $3Km/h$. The rest of the parameters in modules other than 701 are set to default.

5.1 Test Results

The KPI values from module 701 of each simulation is used to calculate the probability of a fault. Using the thresholds previously determined each KPI is assigned a state. Equation 2 is used to rank the possible faults. In this paper

due to space limitation, KPIs for six faults and one normal case were collected on two occasions i.e. there are two test runs for each fault. The results are described below.

The result shows that in the thirteen tests the model had calculated the real fault correctly. Only in one case the most probable cause is not the real fault. The analysis of each test is found below.

- **Test** 1*a* **and** 1*b***:** *Normal – No Fault* In order to develop a real model it is important to run tests in a normal state of the network without any fault. In both tests, *The model*, identifies as the most likely fault, the no fault case. In the first test, this likely is with a 70.5%, and in the second test there is no absolutely doubt about what is the fault because it has a probability higher than 99%. The Table 13 shows the rest of the fault probabilities.

- **Test** 2*a* **and** 2*b***:** *P_{ms} max forced to an incorrect value.* In both tests the probability is 99%. Other possible faults have a low likelihood to be the real fault. This indicates a strong association with the fault.

- **Test** 3*a* **and** 3*b***:** *BLER Target RT Uplink forced to incorrect value.* In test 5 the diagnosis failed as it only assigns 44% to the real fault while the fault *Pot Pilot in minimum value*, is the first in the ranking with a probability of 53%. In test 3b, the diagnosis is correct but it still doesn't have a high probability. This can be interpreted in several ways. On one hand, the thresholds of the main KPIs for this fault are not well defined, i.e. one low value could be classified as normal because the normal range is too wide and although the value is in the border,with the low state, the model assigns it the same probability of a normal case.

- **Test** 4*a* **and** 4*b***:** *BLER Target RT Uplink forced to an incorrect value.* The diagnosis is clear for test 4a and 4b. The fault is correctly identified with an average of $80 - 85\%$ probability. The second highest probability is No Fault, with a 20% and the rest are negligible.

- **Test** 5*a* **and** 5*b***:** *P_{bs} Max forced to an incorrect value* In both test 5a and 5b the fault is diagnosed with a large margin of success.

- **Tests** 6*a* **and** 6*a***:** *Pot Pilot forced to a minimum value (Pot Pilot Min)* The model correctly diagnosis the fault with a high percentage. There are some probabilities that indicate it could be a normal case. However, the likelihood of another fault is almost residual.

- **Tests** 7*a* **and** 7*b***:** *P_{ms} Min MS forced to an incorrect value* In the test P_{bs} Max BS forced to an incorrect value. The model correctly identifies the fault. The probability is high and the diagnosis clearly states in this scenario the fault is P_{ms} Min is incorrect.

6. CONCLUSION

Bayesian Networks have demonstrated themselves to be a powerful tool when applied to troubleshooting, mainly in the fault detection field. It is possible to rank a large number of faults according to their fault probability in abnormal situation in an automated manner. This concept was extended to troubleshoot a UMTS/WLAN test platform. KPIs from the platform were taken for six faults and used in diagnosing the fault. The results show that in thirteen out of fourteen cases the fault was calculated in the first ranking with a high probability of success, higher than 50%. And in six cases out of thirteen the real cause of the scenario were calculated with a probability of 99%. Only in one case, the real fault was ranked as the second most probable fault cause.

The current model can be extended to cover all faults in the testbed. Furthermore, the method presented in this paper can be applied to any test platform that produces indicators. To conclude, the approach of using Bayesian networks serves as an accurate solution to the problems of traditional troubleshooting. The properties of the Bayesian network makes it possible to quickly modify models and thereby overcoming the problem with the variety of component configurations in modern cellular networks.

Acknowledgment

This work was supported by the National Communications Network Research Centre, a Science Foundation Ireland Project, under Grant 03/IN3/1396.

7. REFERENCES

[1] D. Nivoski. *Constructing bayesian network for medical diagnosis from incomplete and partially correct statistics*. *IEEE Trans. Knowledge Data Eng.*, Vol. 12(Number 4):pp. 509–516, 2000.

[2] Franco Taroni, Colin Aitken, Paolo Garbolino, and Alex Biedermann. *Bayesian Networks and Probabilistic Inference in Forensic Science*. Wiley, February 2006.

[3] Z. Altman, B.Solana, R.Khanafer, L.Molsten, R.Barco, L.M. Matamoros, etc. *Automating Diagnosis in Troubleshooting*. Celtic Gandalf Deliverable D5.3a, December 2006.

[4] Pradipta De, Ashish Raniwal, Srikant Sharma, and Tzi cker Chiueh. Design considerations for a multihop wireless network testbed. *IEEE Communications Magazine*, pages 102–109, October 2005.

[5] N. Goldszmidt N. Friedmann. *Learning Bayesian Network from data*, January 1998. http://www.erg.sri.com/people/moises/tutorial.

[6] D. Niedermayer . *An introduction to Bayesian Network and their contemporary Applications*, December 1998. www.niedermayer.ca/papers.

[7] R. Kreher. *UMTS Performance Measurement: A Practical Guide to KPIs for the UTRAN Environment*. John Wiley.

[8] W. Kent, R. Skehill, M. Barry, and S. McGrath. *An Integrated UMTS-WLAN Testbed*. 3rd IEEE International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom), May 2007.

[9] R. Skehill, M. Barry, M. O'Callaghan, N. Gawley, W. Kent, and S. McGrath. *Common RRM Approach to Admission Control for Converged Heterogeneous Wireless Networks*. Special Issue of IEEE Wireless Communications Magazine on Technologies On Future Converged Wireless And Mobility Platform, April 2007.

[10] B. Solana. *Determination of parameters and KPIs adapatation processing*. Celtic Gandalf Deliverable 5.1 A, Telefónica I+D, June 2006. Version:1.0.

Table 4: ***Testbed in Normal Operating Conditions. KPI Results with no Fault Tests***

Run	RT BLER(u)	Active Users	Load Factor(u)	Load Factor(d)	fdL	RT Req/s	RT Rej/s	STR Req/s	STR Rej/s	WWW Req/s	WWW Rej/s
1	0,041	531,06	0,85	0,04	0,21	5,95	4.02	0.21	0.17	1.8	0.96
2	0,041	516,54	0,87	0,06	0.3	6.4	6,11	0,24	0.22	1.79	0.97
3	0,047	550	0,87	0,05	0,27	5.9	4.01	0.22	0.23	1.77	0.94
4	0,047	535	0,83	0,09	0,6	6,41	3,91	0,23	0,16	1,78	0,95
5	0,056	448	0,83	0,03	0,21	5,95	4,08	0,27	0,2	1,82	1,01
6	0,044	510,98	0,86	0,06	0,55	5,71	4,06	0,2	0,15	1,75	1,12
7	0,041	521,6	0,86	0,06	0,32	4,02	6,11	0,24	0,18	1,81	0,9
8	0,057	552	0,78	0,04	6,3	6,37	3,89	0,31	0,24	1,79	0,8
Avg.	0,0467	520,6	0,84	0,053	1,41	6,88	5,632	0,298	0,186	1,79	0,956

Table 5: ***Results of BLER RT Target in minimum value***

Run	BLER(u) RT	LoadF(d)	fdL	REJ/s RT
1	0,004	0,08	0,26	4,29
2	0,004	0,08	0,65	4,26
3	0,003	0,04	0,28	4,41
Avg.	0,0036	0,066	0,39	4,32

Table 6: ***Results of BLER rt Target set to a maximum value***

Run	DUR. (min)	BLER(u) RT	LoadF(d)	fdL	REJ/s
1	44	0,191	0,08	0,46	3,84
2	19	0,22	0,08	0,6	3,63
3	17	0,26	0,09	0,64	3,42
Avg.	26,7	0,223	0,083	0,56	3,63

Table 7: ***Results of P_{ms} Max forced to a minimum value***

Run	BLER(u)	BLER(d)	Active Users	LoadF(u)	LoadF(d)	fdL	Req/s RT
1	0,184	0,94	57,41	0,24	0,09	0,46	11,4
2	0,183	0,909	57,28	0,86	0,02	0,22	11,6
3	0,188	0,946	59,25	0,15	0,02	0,1	12,03
4	0,171	0,91	57	0,24	0,05	0,38	12,18
5	0,172	0,93	57	0,19	0,02	0,03	12,04
6	0,18	0,951	59,96	0,18	0,07	0,13	11,66
7	0,188	0,929	60,4	0,19	0,02	0,03	11,86
Avg.	0,180	0,93	58,3	0,29	0,041	0,19	11,82

Table 8: ***Results of P_{bs} Max forced to a minimum value***

Run	DUR. (min)	Active Users	LoadF(u)	REQ/s RT	REQ/s Stream	REJ/s
1	26	267,93	0,015	12,86	0,32	0,02
2	20	275,3	0,015	13,06	0,39	0,03
3	17	264	0,016	13,19	0,4	0,03
Avg.	21	269,07	0,015	13,04	0,37	0,027

Table 9: ***Results of Pot Pilot forced to a minimum value (P_{bs} min)***

Run	DUR. (min)	BLER(d)	LoadF(d)	fdL RRM	REJ/s RT	REJ/s WWW
1	25	0,008	0,13	3,01	4,03	1,12
2	35	0,03	0,36	25,38	4,15	1,19
3	29	0,006	0,56	29,22	4,3	1,14
Avg.	29,7	0,014	0,35	19,20	4,16	1,15

Table 10: ***Results of Pot Min forced to a maximum value***

Run	DUR.	BLER(u) (min)	BLER(u) RT	Active Users WWW	LoadF(u)	LoadF(d)
1	29	0,012	0,27	131,27	0,99	0,02
2	25	0,014	0,26	137,23	0,98	0,02
3	16	0,013	0,27	123,24	0,97	0,02
Avg.	23,3	0,013	0,266	130,58	0,98	0,02

Table 11: Threshold for Test Platform KPIs

KPI	Threshold				
	Very Low	Low	Normal	High	Very High
Active Users	0-300	300-500	500-550	550-1000	1000-2100
Load Factor (UL)	0.0-0.1	0.1-0.7	0.7-0.87	0.87-0.9	0.9-1.0
Load Factor (DL)	0.0-0.01	0.01-0.04	0.04-0.06	0.06-0.08	0.08-1.0
DFL	0.0-0.1	0.1-0.2	0.2-0.6	0.6-3.0	3.0-100
RT BLER(UL)	0.0-0.08	0.08-0.03	0.03-0.06	0.06-0.19	0.19-1.0
WWW BLER (UL)	0.0-0.0001	0.0001-0.07	0.07-0.09	0.09-0.3	0.3-1.0
RTREQ/s	-	-	0.0-6.0	6.0-8.0	8.0-15.0
RTREJ/s	0.0-3.0	3.0-4.0	4.0-5.0	5.0-7.0	7.0-15.0
WWWREQ/s	0.0-1.0	1.0-1.7	1.7-1.8	1.8-2.0	2.0-50
WWWREJ/s	-	0.0-0.9	0.9-1.0	1.0-1.2	1.2-2.0
STREQ/s	-	-	0.0-0.3	0.3-0.35	0.35-1.0
STREJ/s	-	-	0.0-0.2	0.2-0.25	0.25-0.5

Table 12: ***Fault Values Compared to Normal Values. Faults are Introduced into Module 701.***

Fault	Normal Value	Fault Value
BLER RT MIN	0.01	0.00001
BLER RT MAX	0.04	0.1
P_{bs} MIN	43 dBm	-23 dBm
POT PILOT MIN	33 dBm	-21 dBm
P_{ms} MAX	21 dBm	-23 dBm
P_{ms} MIN	-44 dBm	21 dBm

Table 13: Results

Test 1a:	*No Fault*		**Test 1b:**	*No Fault*
FAULT	Probability %		FAULT	Probability %
No fault	70.50		No fault	99.724
BLERrtUmax	29.22		BLERrtUmax	0.23
Pot Pilot Min	0.251		BLERrtUmin	0.042
BLERrtUmin	0.018		Pot Pilot Min	0.004
Test 2a :	*P_{ms} Max*		**Test 2b :**	*P_{ms} Max*
FAULT	Probability %		FAULT	Probability %
P_{ms} Max	99.97		P_{ms} Max	99.635
P_{ms} Min	0.03		P_{ms} Min	0.358
P_{bs} Max	0.01		P_{ms} Max	0.006
Test 3a :	*BLER RT U min*		**Test 3b :**	*BLER RT U min*
FAULT	Probability %		FAULT	Probability %
Pot Pilot Min	52.245		BLER RT U min	52.71
BLER RT U min	44.755		No fault	29.808
BLERrtUmax	2.453		Pot Pilot Min	10.49
No fault	0.46		P_{ms} Min	5.786
P_{ms} Min	0.076		BLERrtUmax	1.083
Test 4a :	*BLER RT U max*		**Test 4b :**	*BLER RT U max*
FAULT	Probability %		FAULT	Probability %
BLERrtUmax	85.237		BLERrtUmax	80.025
No fault	13.388		No fault	19.936
P_{bs} Max	0.513		BLER RT U min	0.022
Pot Pilot Min	0.484		Pot Pilot Min	0.016
P_{ms} Min	0.25		P_{ms} Max	0.001
Test 5a :	*P_{bs} Max*		**Test 5b :**	*P_{bs} Max*
FAULT	Probability %		FAULT	Probability %
P_{bs} Max	99.996		P_{bs} Max	100
P_{ms} Min	0.004		No fault	0
Test 6a :	*Pot Pilot Min*		**Test 6b :**	*Pot Pilot Min*
FAULT	Probability %		FAULT	Probability %
Pot Pilot Min	78.303		Pot Pilot Min	70.775
No fault	21.581		No fault	28.527
BLERrtUmax	0.105		BLERrtUmax	0.685
BLERrtUmin	0.011		P_{ms} Min MS	0.007
Test 7a :	*P_{ms} Min*		**Test 7b :**	*P_{ms} Min*
FAULT	Probability %		FAULT	Probability %
P_{ms} Min	99.986		P_{bs} Min	100
P_{ms} Max	0.012		No fault	0