Introducing an Unsupervised Automated Solution for Root Cause Diagnosis in Mobile Networks

Maha Mdini, Gwendal Simon, Alberto Blanc, Julien Lecoeuvre

▶ **To cite this version:**

Maha Mdini, Gwendal Simon, Alberto Blanc, Julien Lecoeuvre. Introducing an Unsupervised Automated Solution for Root Cause Diagnosis in Mobile Networks. IEEE Transactions on Network and Service Management, 2019, pp.1-1. 10.1109/TNSM.2019.2954340 . hal-02377865

HAL Id: hal-02377865

https://imt-atlantique.hal.science/hal-02377865

Submitted on 24 Nov 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Introducing an Unsupervised Automated Solution for Root Cause Diagnosis in Mobile Networks

Maha Mdini*†, Gwendal Simon*, Alberto Blanc*, Julien Lecoeuvre†,
*IMT Atlantique †Exfo

***Abstract*—**
Today's network operators strive to create self-healing cellular networks that have a fully automated troubleshooting management process. To this end, the network monitoring system should be capable of detecting issues, diagnosing them, and triggering the adequate recovery action. In this paper, we propose an unsupervised solution to diagnose the root causes of network issues. As monitoring systems collect a large number of logs from the different devices in their networks, it is possible to determine which connections resulted in a poor user experience and apply a failed/successful label. Our solution, Automatic Root Cause Diagnosis (ARCD), analyzes labeled connection logs to identify the major contributors to the network inefficiency (e.g., a faulty core device) as well as the incompatibilities between different elements (e.g., make and model of a phone not being able to access a service). We evaluate the effectiveness of our solution by using logs from three different real cellular networks. In each case, ARCD was able to identify the major contributors and the most widespread incompatibilities. In the three cases, the precision (detection accuracy) and the recall (detection rate) are higher than 90%.

***Index Terms*—Network monitoring, root cause diagnosis, self-healing networks, fault management, data analysis.**

I. Introduction

Cellular networks have become more complex over the years, with multiple co-existing Radio Access Technologies (RATs) from 2G to 4G and now 5G, multiple core network devices from various vendors, multiple services that go beyond regular telephony, and multiple handsets running various Operating Systems (OSs). This growing complexity makes the task of monitoring the network and identifying the cause of performance degradation more challenging for network operators. Most devices in the network generate *log* messages detailing their operations. Based on these messages it is possible to reconstruct what happened in the network, at least to a certain extent. But, given the sheer number and variety of these messages, it is not feasible for human operators to analyze all of them directly. This is why log messages are usually pre-processed before being scrutinized by human experts who are in charge of identifying and mitigating the issues by appropriate actions. This analysis, while partly automated and aided by ad-hoc tools [1], is often time consuming and inefficient. Network operators would like to increase the automation of this analysis, in particular for cellular networks, in order to reduce the time needed to detect, and fix, performance issues and to spot more complicated cases that are not always detected by human operators.

The data that are generated by the monitoring system are a large number of log *entries* (or simply logs), each of them being the report of what happened during a *session*. The term session depends on the specific service: Call Data Record (CDR) for a regular phone call or Session Data Record (SDR) for a service based on Internet Protocol (IP). The log takes usually the form of a series of 2-tuples $(feature, value)$. The feature is a name indicating the nature of the corresponding value (for example cell identifier, content provider, handset manufacturer), while the value is what has been collected for this particular session (in our example, a number that enables to uniquely identify the cell, the name of a provider, and the name of a manufacturer). The root cause of a network malfunction can be either a certain 2-tuple, or a combination of k 2-tuples. Figure 1 shows a simplified view of an LTE cellular network, including some of its elements and the corresponding monitoring system, which collects data from the different devices to produce the CDRs and SDRs.

The literature related to network monitoring is extensive (see Section II). However, identifying the root cause of problems in modern cellular networks is still an open research question due to specific requirements related to the nature of this type of networks: First, a diagnosis system should work on various types of logs (phone, data, multimedia session) because, nowadays, cellular networks carry significant amounts of data traffic as well as voice. Second, a diagnosis solution has to deal with the increasing number of features. Logs can now include features related to the service (e.g., the content provider, the quality and priority classes), to the network (e.g., the RAT and the gateways involved), and to the user (e.g., the handset type and the handset manufacturer). Furthermore, these features can depend on each other due to the architecture of network and services. Third, a diagnosis solution has to address the complex interplay between features. For example, an OS version not supporting a particular service. Both the service and the OS can behave normally in a majority of sessions when scrutinized independently; the issue can only be diagnosed in logs containing both. Finally, the diagnosis solution should focus on problems that have an actual impact on the network performance. A problem that happens *sometimes* in a device that is used by millions of users can have a greater importance than a problem happening *always* in a device used by only a few users. The balance between number of occurrences and inefficiency is a matter of prioritizing mitigation actions.

A. Objectives of our Diagnostic System

Our goal is to design a diagnostic system, which addresses three key and challenging features that network operator

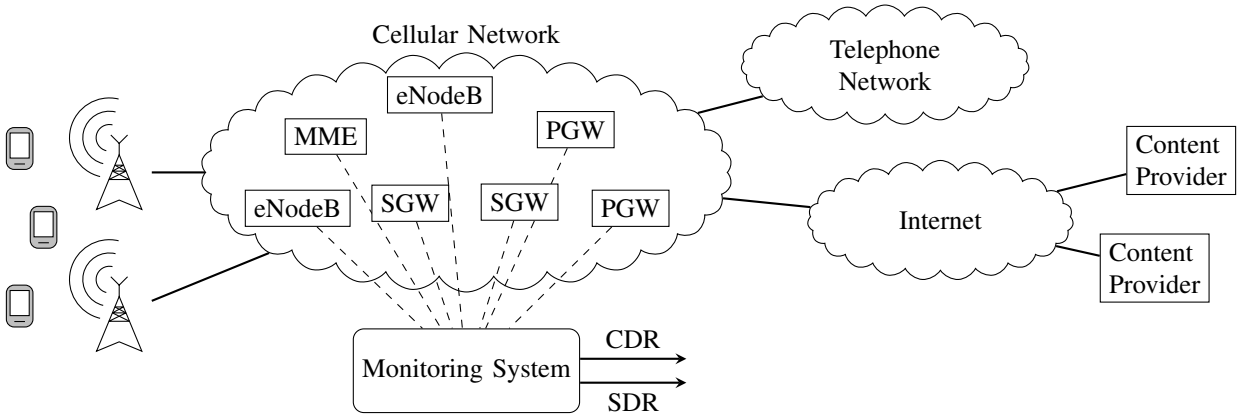

Figure 1: System architecture of an Long Term Evolution (LTE) network with a subset of the monitored elements; Evolved Node B (eNodeB), Packet data network Gateway (PGW), Serving Gateway (SGW), and Mobility Management Entity (MME)

management teams are waiting for:

- **Identifying Major Contributors**: We call major contributors all the elements that cause a significant decrease of the overall network efficiency. For example, a faulty MME causing a large number of dropped calls. Our goal is to identify the major contributors in the whole set of logs. Once these major contributors have been identified, it is possible for human experts to scrutinize them in order to resolve the underlying issues.
- **Detecting Incompatibilities**: A consequence of the great variety of devices involved is that incompatibilities between some of them can also result in poor Quality of Experience (QoE) for the users. An incompatibility is a failing combination of two, otherwise properly working, elements. As previously stated, a new release of an OS can typically be incompatible with a service (e.g., voice calls, toll numbers).
- **Forming Equivalence Classes**: Equivalence classes are sets of key-feature values that correspond to the same underlying problem or that are strongly correlated. For instance, if a cell has only a few users and one of them is significantly more active than all the others (e.g., an automated bot making robocalls) and if the corresponding International Mobile Subscriber Identity (IMSI) has an abnormally high failure rate, the corresponding cell might have an abnormally high failure rate as well, but this is only because the specific IMSI is present in the overwhelming majority of the logs of the calls starting in that cell. We build equivalence classes to prevent the same cause to appear in failing logs under multiple features.

We provide more details in Section IV on the reasons for which these three tasks are relevant and why it is not easy to automate them reliably.

B. Contributions

In this paper, we present ARCD: a complete solution to identify the root cause of network inefficiencies. In a previous paper [2], we have introduced the main idea behind ARCD. In the present paper, we provide a complete and detailed description of the solution. In particular, while our previous paper has only focused on the identification of major contributors, we present here how ARCD manages to detect incompatibilities and equivalence classes. To our knowledge, it is the first time the algorithms behind a complete in-production anomaly detection system for mobile networks are described in an academic paper.

We first present the main algorithms that run the ARCD system. Then, we evaluate ARCD by analyzing the logs of three different cellular network operators. Our results show that with an automated solution, we can not only carry out the analysis done by experts but we can go to a finer level of diagnosis and point to the root causes of issues with high precision.

II. Related Work

The literature on automatic root cause diagnosis is extensive [3]. We distinguish two main approaches. One approach is characterized by analyzing one feature at a time. The second approach is based on dependency analysis.

A. Approaches

Isolated Features. Some researchers consider each feature in isolation (e.g., handset type, cell identifier, service), applying statistical inference, Machine Learning (ML) techniques, or expert rules to identify the elements causing network inefficiency. As an example, Gómez-Andrades et al. [4] use an unsupervised technique based on Self Organizing Maps and Hierarchical Clustering to identify the cells responsible for network inefficiency. Other studies, which are not limited to the radio context, have an end-to-end view of the network considering only one feature at a time. For example, Serrano Garcia et al. [5] propose a dynamic diagnosis framework based on weighted correlation. This framework is applicable at any level of the network (e.g., cells, core equipment). Such an approach has also been explored in contexts other than mobile networks. For instance, Zheng et al. [6] perform rough root cause location on High Performance Computing (HPC) systems based on software logs to classify issues into three classes

of failures: hardware, software and application. This approach, based on analyzing each feature in isolation, can be accurate and easy to understand by end users, but its main drawback is that it does not take into account the dependencies between the features. For instance the cells connected to a malfunctioning Base Station Controller (BSC) may appear as inefficient. The approaches based on considering one feature at a time have also the obvious limitation of ignoring all the problems caused by more than one feature, such as incompatibilities and causal effects. These induced effects cannot be detected unless one uses dependency analysis.

Dependency-Based. Some researchers have focused on hierarchical dependencies resulting from the topology of the network, e.g., the content providers of a mis-configured service having their content undelivered. To identify such dependencies, they rely on the topology of the network, which is not always straightforward to obtain in an automated way. Jin et al. [7] combine multiple classifiers to rank the locations of the issues. Then, they exploit the topology of the wired access network to explain the dependencies between the problems. Mahimkar et al. [8] monitor the Key Performance Indicators (KPIs) to identify the most inefficient elements in the network. Then, they explore the higher level elements in the topological graph of the network to identify the elements impacted by the same problem. By relying on the network topology to identify dependencies, one may miss some relevant occasional dependencies resulting from co-occurrence or coincidence, e.g., a group of cellphone roaming users (tourists) accessing the same cell. These dependencies are not predictable by the experts. To explore both hierarchical and occasional dependencies, different statistical methods have been proposed. These studies, while addressing some of the challenges related to root cause analysis, do not meet all the requirements of a complete and production-ready diagnostic system. First of all, it is non-trivial to automatically and reliably discover the network topology. Solutions that require this as an input are not practical, especially for networks as complex as cellular ones. Second, the aforementioned statistical tools are not well suited to handle logs with many features (more than one hundred in typical LTE networks), especially when many of these features are categorical. Lastly, for a solution to be viable in a production environment, it must also be scalable and present results that are easy to interpret by human operators.

B. Techniques

We can classify root cause diagnosis techniques in two main categories: *expert systems* and *ML*.

Expert Systems. Some researchers used expert systems to perform root cause diagnosis [3, 9]. Starting from a set of known issues, they build rules based on KPIs that can determine the nature and location of each problem. As they are based on known issues, they need be constantly updated as new class o problems are discovered. This is the norm in mobile networks, given the rapid growth of mobile networks and the multiplicity of actors (equipment vendors, handset manufacturers, software companies). If human experts need to implement these updates, the resulting costs can be prohibitive, given the large volumes of data generated by modern networks. This is why there have been recently some efforts to build more autonomous systems based on ML.

Machine Learning. There is a vast literature on classification and prediction, including applications to self-healing cellular networks [10]. Network logs may have Quality of Service (QoS) labels. As an example, a CDR is labeled as successful if the call is established and failed if the call drops. Even though we are dealing with labeled data, our goal is not to predict the label, which is the goal of supervised ML techniques. Our goal is to identify the root causes of problems that can negatively affect the users of cellular networks. Classification, however, is a potential solution. Some researchers applied clustering techniques to cluster cells or another feature into faulty/non faulty one based on specific KPIs [4]. This approach, as explained in Section II-A, has many limitations. Decision trees are another technique that can be used to infer the association rules in a data set [11]. However, it necessitates a database of solved cases, which would be extremely expensive to generate and to keep up-to-date. On the one hand, due to dependencies between features, human experts are not capable of identifying manually all the major contributors and incompatibilities to train a supervised system. In general, ML algorithms can learn from labels but not rules. Reinforcement Learning (RL) addresses this limitation and thus can be applied to create a self-healing cellular network [12]. However, in our use case, RL has two major problems. First, we cannot deploy a solution that acts directly on the mobile network without human intervention. Second, it is not straightforward to identify a reward function for major contributors (and incompatibilities) that can be automatically evaluated.

III. Data Model and Notations

A. Data Records

As detailed in the Third Generation Partnership Project (3GPP) specification [13], network operators collect data records to report every mobile communication that is established in the network. A data record contains the technical details of a mobile communication without including its content. These records can be used in network troubleshooting [9]. We call *log* an entry in the data records. A log is a series of 2-tuples $(feature, value)$ where the features can be:

Service related extracted via a Deep Packet Inspection (DPI) module such as service type, Mobile Network Code (MNC), content provider, QoS Class Identifier (QCI).

Network related such as RAT, MME, Radio Network Controller (RNC), cell.

User related such as IMSI, handset type.

In a log, every feature is associated with a value. We show in Table I three logs with a few features. Note that every value used in the paper appears with a hashed value to preserve anonymity. Logs from the same cell (logs 0 and 2) or from the same service (0 and 1) can be tracked. The table shows as well that each log has a feature called *label*, which is a binary value indicating whether the communication was satisfactory or not. The label can be either collected directly

by the monitoring system, or it can be computed during a post-processing analysis based on the values of the log.

	first_cell	imsi	tac	service	interface	label
0	a3d2	97c8	c567	ea52	eccb	failed
1	b37a	56ed	ce31	ea52	19c4	successful
2	a3d2	fa3e	c41e	c98e	f487	successful

Table I: Example of a Data Record with a few features

We consider two types of Data Records in this paper:

CDR the records of voice calls. Each time a subscriber attempts to make a call, a CDR is created. If the call is dropped, the CDR is labeled as failed.

SDR the records created to track every Internet connection in cellular networks. An SDR is created each time a subscriber attempts to use an online mobile application. SDRs are often the summary of multiple Transmission Control Protocol (TCP) connections initiated by the mobile application. Unlike CDRs, SDRs are not labeled. However, it is possible to estimate the QoE for the user and thus to attribute a label, based on the data rate, response time, and retransmission ratio.

Thanks to the success/fail label, it is possible to compute the network *inefficiency*, that is the proportion of failed communications. This proportion can be computed over all the logs, or it can be relative to a specific subset (e.g., all the logs that share a given feature-value pair).

B. Notation

Let E be a set of logs and $f_1, f_2, ..f_n$ be the features of the logs. A log $x \in E$ can also be represented as a vector $x = (x_1, x_2, \ldots, x_n)$ where x_i is the value of the feature f_i. Since every log has a label, we can partition E in two disjoint subsets: S containing the logs labelled as successful and F containing the logs labelled as unsatisfactory (i.e., failed).

We introduce the notion of *signature* to group the logs that have certain similarities. A k-signature s is the set of all logs, irrespective of their label, where k pre-determined features $\{f_{p_1}, f_{p_2}, \ldots, f_{p_k}\}$ $(1 \leq p_i \leq n, \forall i)$ have k specific values $\{s_{p_1}, s_{p_2}, \ldots, s_{p_k}\}$. We call the parameter k the order of the signature.

For instance, a 2-signature s that groups all logs from cell *ab34* and a mobile phone running the OS *b4e8* can be represented as:

$$\left(\left(\text{first cell}, \text{ab34}\right), \left(\text{handset os}, \text{b4e8}\right)\right)$$

We denote by $E(s)$ the set of all logs matching the signature s, regardless of their labels. Similarly, we denote by $S(s)$ (respectively $F(s)$) the set of all successful (respectively failed) logs matching signature s.

We define a few quantities that are useful to characterize signatures. (The operator $|.|$ denotes the cardinality of a set.)

Signature Proportion (π): the proportion of logs matching s:

$$\pi(s) = \frac{|E(s)|}{|E|}$$

Complementary Signature Proportion ($\overline{\pi}$): the proportion of logs that do not match s:

$$\overline{\pi}(s) = 1 - \pi(s)$$

Failure Ratio (λ): the proportion of failed logs among those matching s:

$$\lambda(s) = \frac{|F(s)|}{|E(s)|}$$

Complementary Failure Ratio ($\overline{\lambda}$): the proportion of failed logs in the data set without considering the logs matching s:

$$\overline{\lambda}(s) = \frac{|F| - |F(s)|}{|E| - |E(s)|}$$

IV. Objectives of the Diagnostic System

Modern cellular networks are complex systems, containing several devices of different types. Network operators run extensive monitoring systems to keep a constant watch on all these devices, in order to detect misbehaving and faulty ones. Faulty devices are not the only element to influence the QoE of the end-users. For instance, roaming users could be prevented from accessing the network because a misconfiguration of an MME. Users often want to communicate with people (or services) hosted on networks other than the one of the cellular one they are connected to. Because of this, problems in other networks can result in a poor QoE for some of the users of a given cellular network.

A. Major Contributors

One of the consequences of the ever increasing usage of cellular networks is that, even though the overwhelming majority of users have an acceptable QoE, there is no shortage of failed logs in most production networks. Operators would like to be able to identify as quickly as possible, and as efficiently as possible, the major contributors, that is the feature-value pairs (or their combination) corresponding to elements causing a significant decrease of the network efficiency. When we exclude the logs matching these feature-value pairs and recompute the network efficiency, we notice an important increase. This fact means that these feature-value pairs are the ones responsible for the network inefficiency. In other terms, the elements pointed by these feature-value pairs are the root causes of network major issues.

One has to be careful when applying the definition of major contributors, as it can lead to some undesirable results. First, because there is a inherent hierarchy in the cellular network, one can significantly increase the efficiency of the network by excluding all the logs that share a very popular feature-value pair. For instance, we could notice a drastic increase of the efficiency after excluding the logs matching a device in the core network. This does not necessarily mean that core network element is deficient but rather it is due to the fact that the logs matching the core network element cover multiple issues at different levels (cell and user issues).

Second, some elements appear in a statistically significant number of logs, have a high failure ratio, but their inefficiency is *extrinsic*, i.e., caused by other elements. For example, a BSC

connected to four cells can be involved in a large number of calls. If two or three of its connected cells are faulty, excluding all the logs referencing this BSC would result in a larger increase in the efficiency than excluding only the logs containing one of the cells. But we are interested in identifying the faulty cells rather than the functioning BSC.

Third, some elements are highly inefficient (they fail often), however they do not appear in a large number of logs. For instance a subscriber can attempt to make a single call, which is dropped. The inefficiency of the IMSI of the subscriber is 1.0 however it is calculated on only one log. This IMSI cannot be considered as a major contributor since its removal has no visible impact on the overall inefficiency. This example illustrates the tension between major contributors and *weak signals*, i.e., elements that have a high inefficiency but that are shared by a small number of logs.

B. Incompatibilities

While major contributors can potentially affect a non-negligible number of users, incompatibilities often affect fewer users, making them a less urgent issue. For example, even though voice calls using a given technology (4G or 2G) have a very low failure rate, calls started with 4G and then switched to 2G may have an abnormally high failure rate. The reason behind this incompatibility may be an incorrect configuration of the Circuit Switched Fallback (CSFB) procedure or the use of devices from different vendors. Similarly, a new release of a smart phone OS could not be compatible with the Transport Layer Security (TLS) implementation used by a certain content provider, because of a bug in the implementation of the TLS handshake in the new release.

ARCD detects as incompatibilities all the cases where a combination of two otherwise working elements results in a higher failure rate. Certain cases that fall within this category do not necessarily conform with the definition of the term "incompatibility." For example, consider two neighboring cells that work correctly but such that there is a poorly covered area between the two, so that all the calls started in one cell and then handed over to the other experience a high drop rate. ARCD detects this as an incompatibility, even though, there are no two incompatible devices here. This is, nonetheless, an issue that must be detected and addressed in order to increase customer satisfaction.

It is extremely hard for human operators to detect incompatibilities simply by looking at the logs, unless these affect a significant number of logs. This is why network operators need automated solutions that can identify them with little or no human intervention. As we discuss in Section VII-E3, detecting incompatibilities is more computationally intensive than detecting major contributors, this is why ARCD addresses each problem separately, giving operators the freedom to decide how often to trigger each operation.

C. Equivalence Classes

ARCD detects both types of anomaly (major contributors and incompatibilities) as a set of one, or more, feature-value pair(s). Because logs are meant to accurately represent the widest possible number of cases, they contain between 30 and 60 different features (see Table II for the precise number of features present in each data set). It is common to see a strong correlation between one or more features for a set of logs. For example, all SDR corresponding to a given service (feature *service provider*) can have the same value for the IP address of the service (feature *host*), whenever the service provider exposes a single IP address as it does happen for smaller service providers. In this specific example, there is a perfect correlation between the two features, making them completely equivalent for a subset of the logs.

The correlation can also reflect the hierarchy of the cellular network. In the example mentioned above, about a single failing IMSI in a cell with few users, the IMSI feature is not exactly equivalent to the cell-id feature, but, for the purpose of identifying the underlying problem, they are indeed equivalent.

It is important for ARCD to be able to detect these equivalence classes automatically, in order to present them to the human operators, who could otherwise be overwhelmed by a long unorganized list of feature-value pairs. This is why we build a directed graph where each node contains one or more feature value pair. Such a data structure has the added advantage of being well suited to build interactive visualizations allowing users to easily navigate between the different problems and to drill down into the details of each one when needed.

V. Major Contributor Detection

We now explain how ARCD processes data records to detect major contributors and how it then filters the results to produce a graph of dependencies between issues occurring in the network. The first step is to label each log as successful or failed, if the input logs are not already labeled. Then, it identifies the top signatures responsible for the network inefficiency. These signatures are then classified into equivalence classes, which are groups of signatures corresponding to the same problem. Then it generates a graph outlining the dependencies between all the problems. It finishes by pruning the graph to remove unnecessary nodes denoting false problems (elements appearing as inefficient because they share a part of their logs with malfunctioning ones). Figure 2 gives a graphical representation of these steps, which are detailed below.

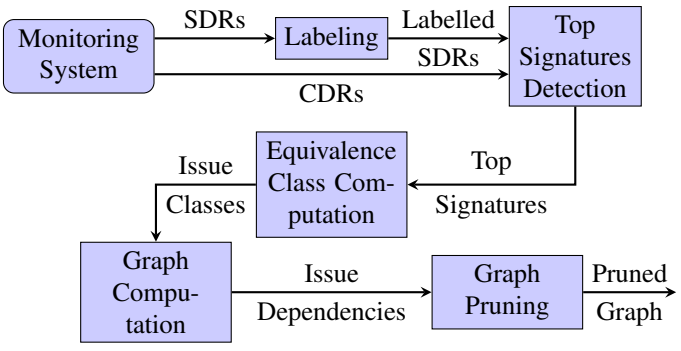

Figure 2: Major contributor detection steps

A. Labeling

The first step consists in labeling the logs. If the data has no success/failure label, we create a binary feature based on standardized criteria specified by the 3GPP. In the case of CDRs, we have a label success/failure based on the Session Initiation Protocol (SIP) messages exchanged between network devices. In the case of SDRs, we assess the QoS of TCP connections based on metrics such as mobile response time, server response time and re-transmission ratio. For each metric, we set a lower bound for an acceptable QoS. An SDR with at least one value below the threshold is labeled as failed.

B. Top Signature Detection

The second step consists in identifying the top 1-signatures (k-signatures where $k = 1$) contributing to the overall inefficiency of the network. To do so, we start by generating the set of all 1-signatures, that is the set of all possible values taken by each one of the features. Then, for each signature we compute two values: the Complementary Signature Proportion $\overline{\pi}$ and the Complementary Failure Ratio $\overline{\lambda}$. The 1-signatures with the smallest values of $\overline{\lambda}$ correspond to the major contributors: removing all the logs belonging to these signatures results in smallest overall failure ratio for the remaining logs. Some of these signatures match a significant fraction of the logs in the system, for instance because the 1-signature corresponds to a device that handles a lot of traffic with a slightly higher failure ratio than the rest of the network.

There is a trade-off between inefficiency and representativeness. The Complementary Signature Proportion ($\overline{\pi}$) indicates whether a 1-signature matters. The larger is $\overline{\pi}(s)$, the less common is the signature s. Our trade-off is thus as follows: on the one hand we want signatures with the smallest values of $\overline{\lambda}$ but not if the corresponding $\overline{\pi}$ is too small. We achieve this goal by maximizing a linear combination of these two values:

$$\nu(s) = \overline{\pi}(s) - \alpha\overline{\lambda}(s)$$

where α is a parameter to weigh the two aspects mentioned above. Large values of α correspond to the "major contributors" (matching many logs), while small values make the focus on "weak signal", i.e., signatures with fewer matching logs but whose failure rate is high. To have a robust solution, we use several values of α: ten values between 0 and 1 $(0.1, 0.2, 0.3, \ldots, 1)$ and then twenty values between 1 and 20 $(1, 2, 3, \ldots, 20)$. The first set of values ($\alpha < 1$) corresponds to the weak signals while the second corresponds to the major contributors.

For each one of these values of α, we compute ν for each 1-signature and we take the twenty signatures with the largest values of ν ("top twenty"). We then compute how many times one of these signatures is in a top twenty. A signature that appears often in the top twenty corresponds to a potential problem. In all our data sets, it is indeed the case that several signatures appear multiple times in the different top twenties. We complete this step by taking the fifty signatures that appear more often in the top twenty. However, we cannot stop here because some of these 1-signatures could correspond to the same underlying problem. That is what the following step address.

C. Equivalence Class Computation

This step consists in grouping signatures related to the same problem. As an example, consider a user connecting to a cell, where he is the only active user, with an uncommon handset type. If, for some reason, the user experiences many consecutive bad sessions, the resulting logs are labeled as failed. In this case, the corresponding IMSI, handset type, and the cell id appear at the top of the signature list that we generated in the previous step. The three signatures point to the same problem rather than three separate problems and have to be grouped into one 3-signature. In general, two signatures are equivalent when they match the same logs. As an aside, we cannot determine the causal relationship between the features and the failure. In our example, either the phone type, the IMSI, the cell or any combination of these three could be the cause of the failure.

We address this problem by computing two values for each pair of 1-signatures in the list produced by the previous step:

$$c_1 = \frac{|E(s_1) \cap E(s_2)|}{|E(s_1)|}$$
$$c_2 = \frac{|E(s_1) \cap E(s_2)|}{|E(s_2)|}.$$

If both c_1 and c_2 are larger than a threshold γ, we consider the two signatures as being *equivalent* and we merge them into one 2-signature. We keep repeating this process as long as at least two signatures satisfy these conditions. This process can generate "longer" signatures as it keeps merging shorter ones. For example, it can crate a 3-signature by merging a 2-signature with a 1-signature, and so on. We stop iterating when no more signatures are merged. In remainder of this paper, we set $\gamma = 0.9$. The outcome of this step is a set of equivalence classes, where each class denotes one problem.

D. Graph Computation

A *hierarchical* dependency is another case of multiple signatures corresponding to the same underlying problem. For instance, a BSC connected to faulty cells would appear as inefficient event if it is not the cause of the problem. In order to highlight this type of dependencies, we create a graph to model the dependencies between equivalence classes created in the previous step. Each equivalence class can be seen as a k-signature. Equivalence classes are presented as the nodes of the graph. To connect the nodes we need to test one way dependencies between equivalence classes (since we have already dealt with mutual dependencies to identify equivalence classes). Therefore, for each k-signature s_1, we find the all the signatures s_2 such that:

$$\frac{|E(s_1) \cap E(s_2)|}{|E(s_1)|} > \gamma$$

which means that the logs matching s_1 are approximately a subset of the logs matching s_2. This way, we find all the parent

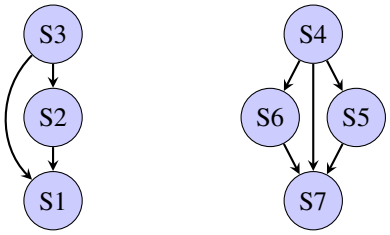

Figure 3: Multiple Path Examples

nodes of s_1. The output of this process is an acyclic direct graph, which is not necessarily connected.

The graph may have superfluous connections, as shown in the example with three signatures s_1,s_2,s_3 in Figure 3. If s_1 depends on s_2 (s_2 is the parent node s_1) and s_2 depends on s_3 then s_1 depends on s_3. If we generate the graph as explained above, we have two paths between s_1 and s_3: a direct connection and another connection via s_2. In this case, the direct connection between s_1 and s_3 is irrelevant since it does not add any information compared to the connection via s_2. To solve this problem, we use the Depth-first Search algorithm to find all the paths between *every pair of connected nodes* and then we keep only the longest path. Note that keeping only the longest path does not lead to any information loss.

Consider the second case in Figure 3. There are three paths from s_4 to s_7. Without loss of generality, assume that we have considered the path through s_6 as the longest one. Then, we will keep only this path between s_4 and s_7. But, since there is only one path between s_4 and s_5, this link is kept as the longest one between s_4 and s_5. The same argument applies to the link between s_5 and s_7. In this case, we end up removing only the link between s_4 and s_7.

E. Graph Pruning

The structure of the graph allows the exploration of faulty devices and services in a hierarchical way: At the top, we find frequent signature (having a high π) such as core network equipment, popular services and handset manufacturers. At the bottom of the graph, we have less frequent signatures such as user IMSIs, host IP addresses, and the least used cell ids.

In a well-constructed graph, we need each child node to have extra information compared to its parent nodes. Otherwise, it is irrelevant and it should be removed. In our case, we know that the parent node is inefficient to some extent (all the nodes in the graph are made up of the inefficient signatures selected in step 1). And, as the child node matches a subset of logs of the parent, it is expected to be inefficient as well. Therefore, presenting the child node is meaningful only in the case where it is more inefficient than at least one of its parent nodes. To remove superfluous nodes, we define a measure called *Relative Failure Ratio* λ_r. Suppose that we have two connected nodes (parent and child). λ_r is defined as follows:

$$\lambda_r(s_c, s_p) = \frac{\lambda(s_c) - \lambda(s_p)}{\lambda(s_p)}$$

where s_p is the signature in the parent node and s_c the one in the child node. For each node, we calculate its relative failure ratio with respect to all its parents. We keep the node if at least one of the relative failure ratios is greater than 0.01. Otherwise, we remove it. Every time we remove a node, we connect its ancestors to its successors. After this pruning operation, every child node in the graph is more inefficient than at least one of its parent nodes. In such case, we have two possible scenarios:

In the first one, the child node presents a separate problem. This could be the case of a user trying continuously to call an unreachable number through a cell having an interference problem. In the graph, we can find the node containing the user IMSI as a child of cell id node with user IMSI failure ratio being higher than the cell failure ratio. The user calling an unreachable number and the radio interference problem are two separate issues. It is therefore important to keep the two nodes.

In the second one, the child node is the root of the inefficiency of the parent node. Consider the case of a large group of roaming user (tourists, for example) accessing the network through a small cell (with few resident users). The roaming users may experience a bad QoE because of a roaming issue between their home network and the host network. Since the roaming users are the main users of the cell, the cell has a high failure ratio λ. In the graph, we find the MNC of the roaming users as a child node of the cell with a slightly higher λ. In this case, the node containing the cell id has to be removed since roaming is the real issue.

To deal with the second scenario, we proceed as follows. Consider two connected nodes: a parent node s_p and a child node s_c. Let λ_n be the overall failure ratio of the network.

$$\lambda_n = \frac{|F|}{|E|}$$

Our goal is to know whether the high failure ratio of s_p is due to s_c. To do so, we restrict to the logs matching s_p, $E(s_p)$ instead of using the whole data set. In this subset we calculate the complementary failure ratio $\overline{\lambda}(s_c)$ which is the failure ratio of s_p after removing the logs matching s_c. If $\overline{\lambda}(s_c) \leq \lambda_n$, then s_p is a non faulty signature and the parent node is removed. Otherwise, we are in the first scenario and the two nodes present two different problems. As previously mentioned, each time we remove a node, we connect its ancestors to its successors. With these new connections, we may have other nodes to remove. We repeat the pruning process as long as it has removed at least one node in the previous iteration. This process is guaranteed to terminate as it only removes nodes.

VI. Incompatibility Detection

The procedure described in Section V allows us to detect major contributors that have a high impact on the network inefficiency. In this section, we explain how ARCD finds incompatibilities. By definition, these involve more than one feature-value pair making their detection more computationally demanding than major contributors. We only needed to compute all the 1-signatures for major contributors, that is the set of all logs having the same feature-value pair, for

all the different feature-value pair present in the logs. When looking for incompatibilities, we compute all the 2-signatures that is the set of all logs having the same two feature-value for two features, for all the features and their values in the logs. Because of this, finding incompatibilities requires more memory and more processing time. We decided to separate the two problems so that operators can decide when to trigger each detection separately, depending on their needs and the availability of computing resources.

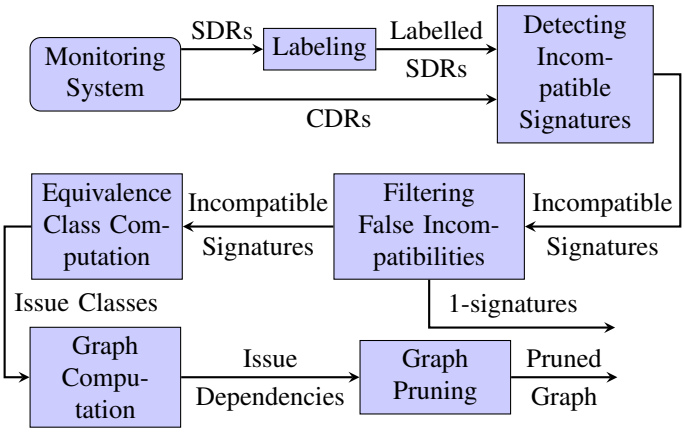

Figure 4: Incompatibility detection steps

Figure 4 describes the main steps of incompatibility detection. We start by labeling data as explained in Section V-A. Then we identify the signatures pointing to incompatibilities throughout the network. Next, we filter false incompatibilities, that is cases where the inefficiency in the logs sharing a given 2-signature is actually explained by a third faulty element that is also present in most of the logs of the 2-signature in question. Finally, similarly to Section V, we compute the equivalence classes.

A. Identifying incompatible signatures

The goal of this step is to find the signatures corresponding to incompatible elements. To do so, we start by generating the sets of all 1-signatures and 2-signatures. For each signature, we measure the failure ratio λ. To decide whether a 2-signature points an incompatibility, we proceed as follows.

Let us consider a 2-signature $s = \{f_i = v_i, f_j = v_j\}$ where v_i and v_j are values taken by the features f_i and f_j, corresponding to two 1-signatures ($s_i = \{f_i = v_i\}$ and $s_j = \{f_j = v_j\}$ respectively). For example, we could have:

$$s_i = (\text{service}, \text{a587}) \quad s_j = (\text{handset os}, \text{c255})$$
$$s = ((\text{service}, \text{a587}), (\text{handset os}, \text{c255})).$$

To determine whether two feature-value pairs are incompatible, we compute the *gain* of s, defined as:

$$g(s) = \lambda(s) - \max(\lambda(s_i), \lambda(s_j)).$$

The gain allows us to evaluate the impact of combining two 1-signatures on the failure ratio. In other words, we check if the combination of two elements is more inefficient than each one apart. If the gain is larger than a threshold, set to 0.2 in our implementation, we consider that the 2-signature s corresponds to incompatible elements.

B. Filtering False Incompatibilities

A combination with a higher failure ratio than each of its components does not imply automatically that we are dealing with an incompatibility. For example, consider the service type Instant Messaging (IM) and a specific content provider, both having a low failure rate. However, the combination of these two elements has a high failure rate. This combination could be identified in the previous step as an incompatibility. However, by studying the logs matching this combination, we may find a single host delivering IM service in the content provider network. The root of the issue, here, is not an incompatibility between IM and the content provider but rather a malfunctioning host.

In order to filter such cases, we proceed as follows. For each 2-signature selected in the previous step (s), we identify its matching set of logs $E(s)$. In this subset, we find all the 1-signatures t that represent a significant fraction of $E(s)$:

$$\tau(s) = \left\{ t | \frac{|E(t) \cap E(s)|}{|E(s)|} > \gamma \right\}$$

where $\gamma = 0.9$. Then, we re-compute the gain:

$$g(s) = \lambda(s) - \max_{t \in \tau}(\lambda(t)).$$

If the gain remains higher than the threshold (0.2), it means that the combination is a real incompatibility: There is no third element with a high failure ratio co-occurring with the combination. Otherwise, if the gain drops below the threshold, there is a third element that is at the origin of the high failure ratio rather than the combination of the two elements. In this case, the issue is not an incompatibility but a highly inefficient single element. We report this element separately with the appropriate 1-signature and we discard the corresponding 2-signature from the incompatibility list.

C. Equivalence Class Computation

At this point, we have a list of 2-signatures corresponding to incompatibilities. But some of the signatures might actually correspond to the same underlying issue. For instance, suppose that we detected an incompatibility between a handset type and an eNodeB identifier. Suppose as well that this eNodeB has only one static IP address so that we could have detected the same incompatibility twice. In this case we should group the eNodeB identifier and its IP address as an equivalence class that is incompatible with the handset type. The goal of this step is to find for each feature-value pair present in one of the 2-signatures, the classes of elements that are incompatible with it.

Let Ω be the set of all 2-signatures that have passed the filtering step, where each 2-signature (ω_i) is composed of two 1-signatures β_i, δ_i. For each one of these constituent 1-signatures we build a list of all the other 1-signatures with which it is incompatible. In other words, for every 1-signature (ρ) in the union of all the β_i and δ_i we compute $I(\rho) = \{\theta | (\rho, \theta) \in \Omega \text{ or } (\theta, \rho) \in \Omega\}$. As $I(\rho)$ is a set of 1-signatures, we can proceed as in Section V-C to group $I(\rho)$ into equivalence classes. Recall that this process can create "longer" signatures (2-signatures, 3-signatures, etc.) so that,

at the end of this process, for each 1-signature ρ, which was in one of the original 2-signatures in Ω, we have a set of k-signatures that are incompatible with it.

D. Graph Computation

Some incompatibilities may result from other ones. For example, if a a service and an OS are incompatible, all the OS versions will be incompatible with that same service. That is why, to identify the root incompatibility, we need to explore hierarchical dependencies. Therefore, for each signature ρ we create a graph as we did in Section V-D. Note that we have $2|\Omega|$ graphs, that is a graph for each 1-signature ρ.

E. Pruning

At this point of the analysis, for each 1-signature ρ, we have a graph of the elements incompatible with ρ. For each pair of connected nodes in this graph, either the parent (θ_p) is the origin of the incompatibility with ρ or the child (θ_c) is the origin. We proceed as in Section V-E to distinguish between the two cases substituting the failure ratio λ with the gain g. In other words, we check if θ_p (the parent of θ_c) remains incompatible with ρ after removing the logs matching the child (θ_c): we remove the logs matching θ_c; then, we compute the gain of the incompatibility between θ_p and ρ. If the gain is higher than the threshold, the parent θ_p is incompatible with ρ independently of the child θ_c. The incompatibility of the child θ_c with ρ could be seen as a result (inheritance) of the incompatibility of the parent θ_p with ρ. In this case, we remove θ_c from the graph. If, on the contrary, the gain drops below the threshold, this means that the child θ_c is the origin of the incompatibility with ρ. As the logs of the child θ_c are a subset of the logs of the parent θ_p, θ_p appeared as incompatible with ρ. In this case, we remove θ_p from the graph. Every time a node is removed, its ancestors are connected to its successors. We iterate the pruning until we have only isolated nodes, which correspond to the signatures incompatible with ρ. Note that we execute the whole process for *each one* of the constituent 1-signatures identified in the previous step.

VII. Evaluation

We have implemented ARCD in a big data cluster and ran major contributor detection and incompatibility detection as Spark jobs scheduled by the task manager OAR[1] as shown in Figure 5. OAR is a batch scheduler used in resource management in HPC and big data structures [14]. ARCD results are displayed in a Graphical User Interface (GUI). We evaluate the effectiveness and performance of ARCD thanks to three different data sets, collected from three different live cellular networks and stored in an Hadoop Distributed File System (HDFS) file system. In the this section, after briefly presenting each data set, we analyze the major contributor detection results followed by the incompatibilities results. Finally, we compare the performance of ARCD with Learning from Examples Module, version 2 (LEM2), which is a well known algorithm for deducting decision rules from data. We also compare ARCD with Distalyzer [15] which is a log-based root cause diagnosis solution.

[1]http://oar.imag.fr/docs/2.5/OAR-Documentation.pdf

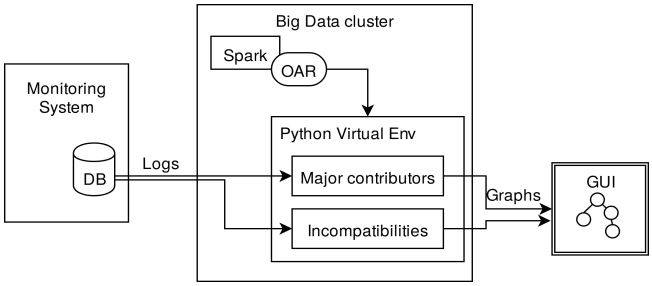

Figure 5: System Architecture

A. Data Sets

We ran ARCD on three data sets from three different operators and anonymized by a hashing algorithm:

Set 1: SDRs recording TCP connections during one hour in a European country. A human expert has cleaned and analyzed this data set so that we can use it as a reference in the validation process. She removed samples with missing and inconsistent fields, and truncated extreme values.

Set 2: SDRs recording TCP connections during one day from another European operator. This set was not examined by an expert (raw set).

Set 3: CDRs logging voice calls during one day from an Asian operator. This is also a raw data set with no pre-specified root causes but where each entry is labeled as successful/failed.

Table II summarizes some of the attributes of the data sets. We used the re-transmission ratio to label each entry for Set 1 and the server response time for Set 2. These metrics were suggested by the experts working with the corresponding operators.

set	number of logs	number of features	number of 1-signatures	failure ratio
1	2500	48	64143	0.05
2	10^7	31	924836	0.18
3	10^6	35	162603	0.08

Table II: Validation Data Sets

B. Expert Validation

We worked with three experts from EXFO, having more than ten years of experience in the domain of mobile network troubleshooting as well as a telecommunication educational background. Each of the experts monitors one of the three networks from which we have extracted the validation data sets. These experts locate and diagnose inefficiency issues in mobile networks as part of their daily tasks using EXFO NOVA solution for network management and optimization.

The experts helped us by providing feedback on the results of our solution. They confirmed that ARCD extracts from

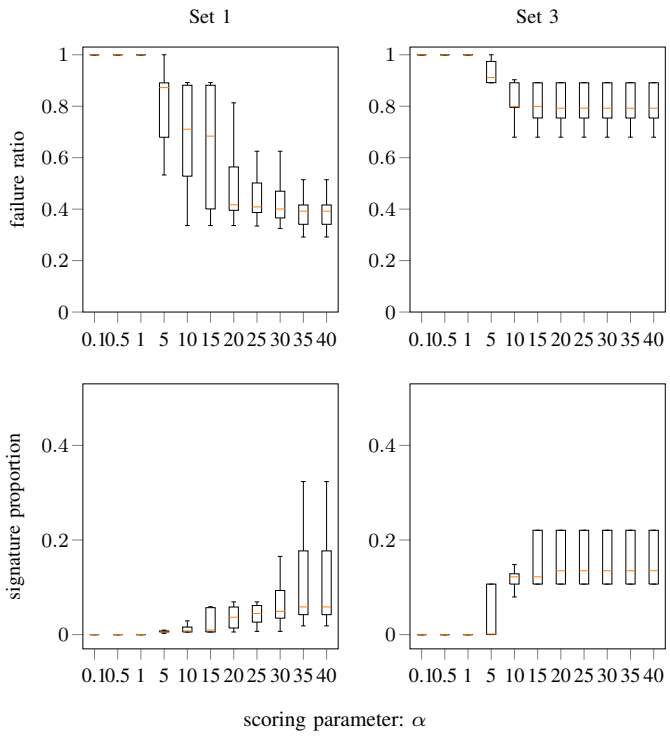

Figure 6: Box-plots of failure ratio and signature proportion as function of scoring parameter for Sets 1 and 3

the logs actionable information for network troubleshooting. They also appreciated the output being presented as graphs that are straightforward to understand, greatly simplifying and streamlining the analysis of the underlying issues and the planning of the mitigation actions.

C. Major Contributor Detection

To validate the results on Set 1, we compared the outcome of our solution with the list of issues identified by human experts. To validate our results on Sets 2 and 3, we have created an expert emulator, which mimics the manual analysis done by human experts. The emulator analyses a limited number of features (less than ten). For Sets 2 and 3, the experts supervising the networks of the corresponding operators have kindly indicated the features they are focusing on. For each feature, the emulator scans the top ten frequent elements (e.g., top popular services). If one element is more inefficient than the global network, the elements is identified as a major contributor. To pinpoint inefficient elements, the expert emulator calculates the following metrics for each element and compares them to the the values of the whole network: failure ratio for CDRs (Set 3); Retransmission ratio (Set1) and server response time (Set 2). These metrics are the same we used in the data labeling phase of ARCD. The concept of hierarchical dependencies is implicitly included in the expert analysis: Experts start with high level elements (e.g., MNC) and then move down to lower levels (e.g., IMSI).

1) Parameter Tuning: As explained in Section V, we used a range of values for α to find the top signatures. Figure 6 shows the distribution of the signature proportion and the failure ratio of the top twenty ranked signatures for each value of the scoring parameter α for Sets 1 and Set 3. As previously explained, the higher α, the higher the signature proportion (corresponding to the most used devices and services). Figure 6 shows also that the smaller α, the higher the failure ratio. This is not surprising as small values of α correspond to the most inefficient elements. As one can notice, for both SDRs and CDRs, the distributions have similar trends. By scanning an interval containing a large range of values for α, we find in practice that we identify the most significant problems, which are feature-value pairs with a sufficiently high number of occurrences to deserves attention and a sufficiently high number of failures suggesting a possible malfunction.

2) Accuracy: To evaluate our solution, we select the following metrics:

True Positives (TP): inefficient elements detected by ARCD and validated either by the expert (Set 1) or by the emulator (Sets 2 and 3).

False Negatives (FN): inefficient elements detected either by the expert (Set 1) or the emulator (Sets 2 and 3) but not detected by ARCD.

False Positives (FP): efficient elements detected by ARCD but not detected in the validation process because their inefficiency is no greater than the overall inefficiency of the network.

Extra Features (EF): inefficient elements detected by ARCD but not detected in the validation process because of the limited number of features analyzed by experts due to time constraints.

Extra Values (EV): inefficient elements detected by ARCD but not detected in the validation process because experts analyze only the top 10 frequent elements of each considered feature.

Precision: The ratio of inefficient elements correctly identified by ARCD (TP+EF+EV) to the total number of elements identified by ARCD (TP+FP+EF+EV).

Recall: The ratio of inefficient elements correctly identified by ARCD (TP+EF+EV) to the total number of inefficient elements (TP+FN+EF+EV).

	Data set	TP	FN	FP	EF	EV	Precision	Recall
ARCD	**Set 1**	11	2	0	38	1	1	0.96
	Set 2	5	2	5	30	10	0.9	0.95
	Set 3	4	1	0	30	16	1	0.98
LEM2	**Set 1**	5	8	7		13	0.72	0.69
	Set 2	7	0	11		43	0.82	1
	Set 3	3	2	1		16	0.95	0.9
Distalyzer	**Set 1**	8	5	0	31	2	1	0.89
	Set 2	4	3	0	17	8	1	0.91
	Set 3	3	2	0	42	4	1	0.96

Table III: Major contributor identification results: A comparison between ARCD, LEM2 and Distalyzer

The first section of Table III shows the overall performance of ARCD, which is satisfying in terms of TPs and FNs. The

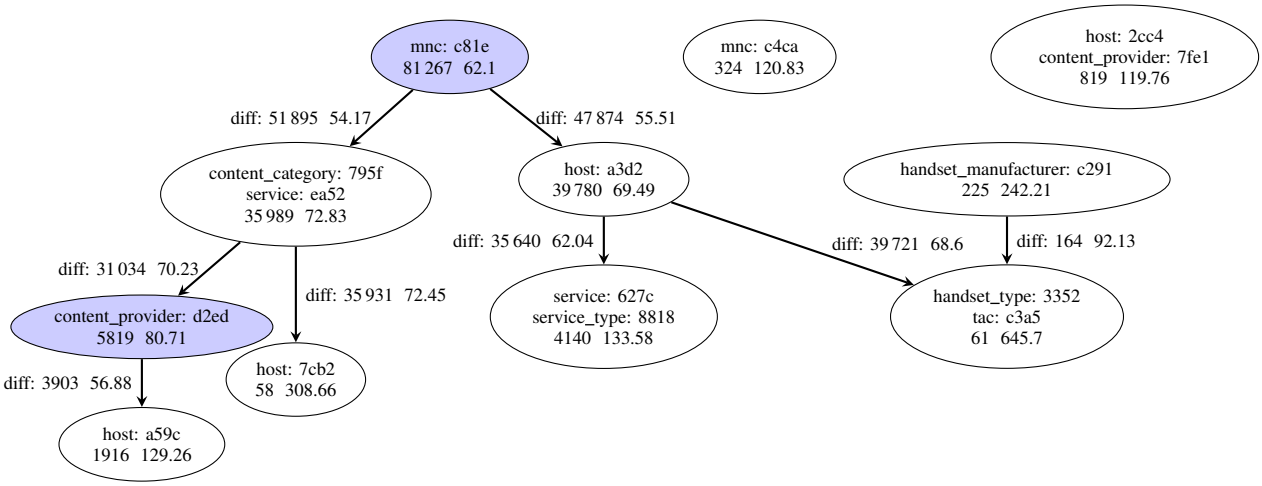

Figure 7: Pruned Graph of Major Contributors for Set 2 (nodes in gray are removed during the pruning process)

interesting aspect of ARCD is its capability to detect issues that are not identified by experts since they focus only on highly frequent elements (such as handset types, services, and core network equipment) due to time constraints. For this reason, they miss issues occurring at a finer level of granularity, which ARCD does detect, such as roaming issues, bad cell coverage and individual users (bots) submitting a large number of call requests to unreachable numbers. The major contributors identified with ARCD explain respectively 91%, 82% and 65% of the failed logs of sets 1, 2 and 3.

3) Output as a Graph: Figure 7 shows an example of the output of ARCD. It is a portion of the graph created based on data from Set 2. The criterion for SDR labeling is the server response time: SDR labeled as failed if response time is larger than 100 ms (the average server response time for the whole network is 60 ms). The nodes contain signatures detected as major contributors. Each node contains the features, a hash of their values (for confidentiality reasons) and two numbers: the number of logs matching the signature and the average response time of the logs matching the signature. We give the response time rather than the failure ratio to ease the interpretation of the graph. The labels on the edges contain the log size and the average response time of the set of logs matching the parent signature and not matching the child signature. The nodes filled in gray are the nodes removed during the pruning process because they denote false problems.

The graph points to two individual problems: a roaming issue (mnc: c4ca); and a content provider issue (host: 2cc4, content_provider: 7fe1). There is also a set of codependent problems with the MNC c81e. This MNC has a large number of logs and a response time slightly higher than the overall network. By detecting this MNC, one may think of a roaming issue. However by removing one of its child nodes, we see that its average response time drops below the average value of the network. That is why this issue was tagged as a false problem and was removed in the pruning step. The same reasoning applies to the Content Provider: d2ed. The node (handset_type: 3352, tac:c3a5) has two parent nodes: (handset_manufacturer: c291) and (host: a3d2). This node is kept in the graph because it is significantly more inefficient than its two parents.

D. Incompatibility Detection

Validating incompatibilities is more complex than validating major contributors. Incompatibilities are fine-grained issues, which experts do not address on a regular basis. While investigating incompatibilities requires a lot of time and effort, the number of impacted subscribers is generally low. To validate our results, we relied on expert help. After testing our solution on the data sets, we presented our results to network management experts who evaluated the accuracy of the results.

1) Accuracy: To evaluate our solution, we use two metrics TP and FP. A TP is an incompatibility detected by ARCD and confirmed by experts to be a real incompatibility, worthy of being reported and investigated. A FP is an incompatibility detected by ARCD whose high inefficiency was actually due to a third feature, detected by the experts. We also measure the precision, which is the ratio of TP to (TP+FP).

Set	TP	FP	Precision
1	10	1	0.91
2	15	2	0.88
3	6	1	0.86

Table IV: Incompatibility results

The overall performance of our solution is satisfying (see Table IV). The issues reported are confirmed by experts to be relevant and deserving attention. In the case of SDRs, we have mainly detected incompatibilities between services/content categories with handset types/OS versions. For CDRs, we have highlighted issues in multi-RAT calls, inaccessible destinations in a specific RAT, and issues with the location update procedure in some cells. The incompatibilities detected by ARCD explain 4%, 16% and 7% of the failed logs of data sets 1, 2 and 3 respectively.

2) Output as a Graph: Figure 8 is an example of part of the output of our solution for the incompatibilities in Set 2. Recall that, for each 1-signature ρ, we generate a graph of the elements incompatible with ρ. Figure 8 is the graph of the

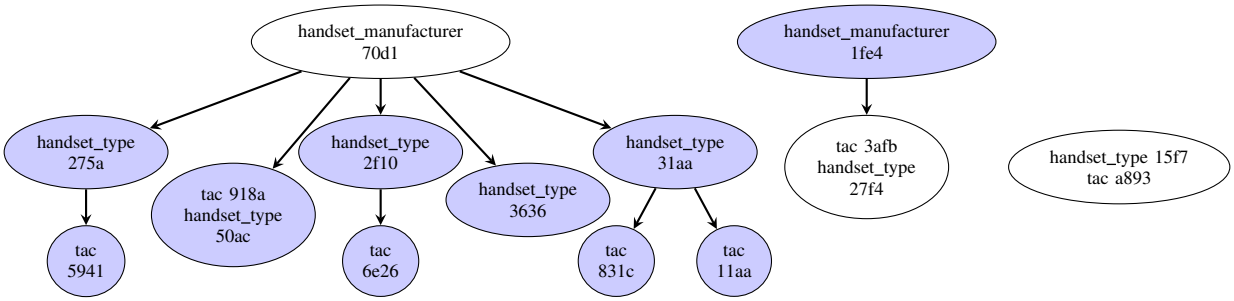

Figure 8: Incompatibilities Pruned Graph for the 1-signature (content_category,795f), gray nodes are removed during the pruning step

elements incompatible with the 1-signature (content category: 795f). The gray nodes are the nodes removed during the pruning step. The issues highlighted here are incompatibilities between a content category and a handset. The graph shows the three families of handsets (three connected components of the graph) that are incompatible with (content category: 795f):

- The first connected component starting with (handset manufacturer,70d1) as incompatible with (content category: 795f): Many handset types and Type Allocation Codes (TACs) of this same handset manufacturer are incompatible with (content category: 795f) as well. These handset types and TACs are removed during the pruning process (colored in gray). The root of the incompatibility with (content category: 795f) is (handset manufacturer: 70d1). The incompatibilities between the handset types and the TACs with (content category: 795f) result from this root incompatibility.
- The two connected nodes (handset manufacturer: 1fe4) and (tac: 3afb, handset type: 27f4): In this case (tac: 3afb, handset type: 27f4) is the origin of the incompatibility with (content category: 795f). The incompatibility between (handset manufacturer: 1fe4) and (content category: 795f) results from it.
- The isolated node (handset type: 15f7, tac: a893): This handset type is simply incompatible with (content category: 795f).

E. Comparison with the LEM2 Algorithm

To evaluate our solution, we compare it with the LEM2 algorithm [16], which is a rule deduction algorithm well suited to our use case: First, it handles inconsistent data, with the same feature-value pairs being sometimes successful and sometimes unsuccessful as in our use case. We aim to detect not only breakdowns ($\lambda = 1$) but also inefficient elements ($0 < \lambda < 1$). Second, it generates a minimal and non-redundant rule set. LEM2 identifies the root of the issues and does not report additional rules related to feature dependencies.

1) LEM2 Results: LEM2 creates decision rules by adding 1-signatures from the most frequent to the least frequent until obtaining a k-signature that does not match any of the successful logs. Once a rule is established, its matching logs are removed from the set of failed logs. Then the same process is iterated until covering all the failed logs. We have made the following two changes[2] to the LEM2 algorithm in order to obtain meaningful results in a reasonable amount of time:

- We stop generating rules when we have explained more than 90% of the failed logs. The idea behind LEM2 is to generate rules from the most important to the least important. As the execution of LEM2 progresses, the rules cover fewer and fewer logs. At the end of the execution, we may end up with a thousand of rules each matching only one log. By adding this breaking condition, we reduce considerably the execution time. We also discard insignificant rules.
- In the process of generating a rule, we stop adding new elements when the rule matches less than 200 logs. Otherwise, the iterations would lead us to have all the rules with high λ but small π. We would detect breakdowns but miss inefficiencies.

rule	count	λ
[(roaming, e4c2), (first_location_type, 0707), (content_provider, 795f)]	48043	0.67
[(service, ea52), (content_provider, 795f), (handset_type, fb38), (src_node_ip_str, 93e3)]	281	0.49
[(service, ea52), (content_provider, 795f), (dest_node_ip_str, a106)]	21926	0.52
[(mcc_mnc, 6364), (roaming, e4c2), (serving_mcc_mnc, 6364), (last_location_type, 0707), (first_location_type, 0707), (dest_node_ip_str, 2f2a), (host, a59c)]	3074	0.92
[(mcc_mnc, 6364), (first_location_type, 0707), (dest_node_ip_str, 79dd), (host, a59c)]	2557	0.90

Table V: LEM2 Example Rules

Table V shows some of the rules that LEM2 found in Set 2. Each rule is a k-signature, for which the failure ratio is high. The count column shows the number of logs matching each rule and the last column gives the failure ratio.

2) Accuracy Comparison: To evaluate the results of the LEM2 algorithm, we used the same metrics as in Section VII-C2. The second section of Table III shows the corresponding results. The last column contains the number extra features and values (EF+EV) that LEM2 finds and that were not detected by the experts. We added EF and EV because some of the LEM2 rules contain both EF and EV.

The numbers of FP and FN are high due to two reasons: first, to deduct the rules, LEM2 conducts an analysis in the set

[2]The modified algorithm is available at https://github.com/mdini/pycon_lem2/blob/master/lem2/lem2_naive.py

of failed logs without comparing the set of failed logs and the set of successful logs, leading to many false positives. Second, during the execution of LEM2, every time a rule is generated, its matching logs are excluded from the analysis. This may be the reason behind false negatives as the sets of logs related to two independent issues may overlap.

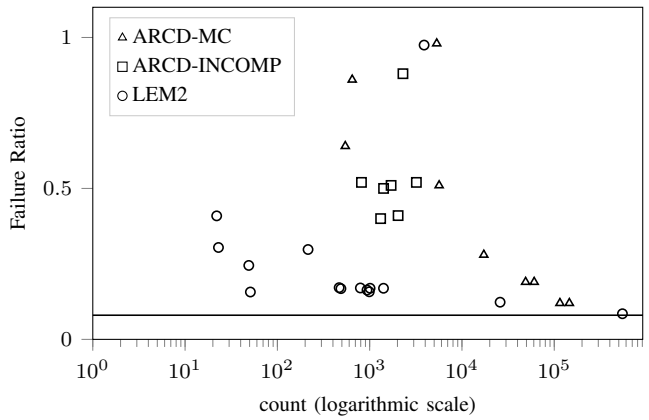

Figure 9: Output from Set 3, signatures identified by ARCD-MC (major contributors), by ARCD-INCOMP (incompatibilities), and LEM2. Each point is a signature, with its failure ratio and the number of its matching logs.

Figure 9 shows the results obtained by ARCD and LEM2 on Set 3. Each point is a k-signature, the x value is the number of logs matching that signature and the y value is its inefficiency. ARCD-MC refers to the signatures detected by the major contributor detection process. ARCD-INCOMP refers to the signatures detected in the process of incompatibility detection. LEM2 refers to the signatures identified by LEM2. The horizontal line shows the inefficiency of the whole network. The figure shows that, in the specific use case of root cause diagnosis in mobile networks, ARCD outperforms LEM2: Most of the signatures identified by ARCD have a higher failure ratio and a higher number of logs compared to LEM2 signatures. We can also observe that the identified incompatibility signatures are more inefficient but less frequent, in general, than the signatures of major contributors.

3) Time Complexity: Let m be the number of features, k the maximum number of value per feature and n the number of logs. As the dependency analysis (equivalence classes, hierarchical dependencies and pruning) is only performed on a limited number of signatures (top fifty), its complexity is then $O(1)$. Thus, the time complexity of major contributor detection in our solution is equal to the complexity of top signature identification which is equal to $O(mkn)$. As $m \ll n$, we can approximate the complexity to $O(kn)$. So, in the worst case, where a feature has a different value for each log, the complexity would be $O(n^2)$. Otherwise, in general, it is approximated to $O(n)$. The same reasoning applies to incompatibility detection. Its time complexity is equal to $O(m^2k^2n)$. In the worst case, it is $O(n^3)$ and otherwise, $O(n)$. The complexity of LEM2 is approximated to $O(n^3)$.

signature	count	λ	$\overline{\lambda}$	t-stat	p-value
(content_provider, 2c11)	474	320	19	59	0
(imsi, 1197)	49	850	25	52	0.00
(server_ip_address, 14f1)	2	200	27	2	0.03

Table VI: Distalyzer Example Results

F. Comparison with DISTALYZER

Distalyzer [15] is a log-based solution for root cause diagnosis. It analyses the dependencies between features and infers issue root causes. Distalyzer comprises four steps. First, it *creates features* from a set of logs. It extracts events and state variables from logs. The second step is the predictive modeling. It *selects the variables* that explain performance issues using T-test. Third, it creates a *dependency network* that summarizes the dependencies between the selected variables. Last, the fourth step, attention focusing, consists on exploring the dependency network to *identify root causes*. Distalyzer identifies major contributors but does not address the question of incompatibilities.

1) Distalyzer Results: To implement Distalyzer, we used Python3. To calculate T-test, we used Scipy Python package. For each extracted 1-signature, we compared the distributions of performance metrics between the set of failed logs and the set of successful logs. As performance metrics, we used the failure ratio for Set 3, the re-transmission ratio for Set 1, and the server response time for Set 2, which are the same metrics used to evaluate ARCD. To create Bayesian dependency networks, we used PGMPY Python package.

In this section and for the sake of simplicity, all the performance metrics are denoted as λ. The mean of a performance metric in the set matching a signature s is denoted $\lambda(s)$. In the same way, the mean of a performance metric in the set not matching a signature s is denoted $\overline{\lambda}(s)$. Table VI shows few examples of Distalyzer application to Set 1. The count column contains the number of logs matching each signature. The t-stat and the p-value columns contain respectively the T-statistic and the p-value calculated when applying the T-test. Table VI shows that the T-test is a good trade-off between the number of occurrences and the performance metric. However, for a high performance metric value, it may prioritize non-critical issues. For example, (imsi, 1197) has a high t-statistic whereas it matches only 49 SDRs. This is explained by its high mean response time: 850 ms.

2) Accuracy and Time Complexity: The third section of Table III contains the results of applying Distalyzer to the three data sets. This table shows that the precision of Distalyzer is equal to 1 which corresponds to no FPs on the three data sets. This can be explained as follows. As the sign of the T-statistic is equal to the sign of $(\lambda - \overline{\lambda})$, the signatures with a positive T-statistic are the signatures having a higher failure ratio (response time or re-transmission ratio) than the rest of the network. Therefore, they are either TPs, EFs or EVs. For this reason, we have added a threshold equal to 0 to Distalyzer to make sure we select only signatures with positive T-statistic (less efficient than the overall network). Distalyzer identifies many EFs and EVs. However, it has few FNs. This is due to

the fact that the T-statistic is proportional to $(\lambda - \overline{\lambda})$. Thus, a signature matching few logs and having an extremely high failure ratio may have a higher T-statistic than a signature matching a large set of logs and having a moderately high failure ratio. Thus, the selected signatures in the top of the ranking may contain many small issues (users, small cells). This can lead to missing issues with large extent in the selected signatures. Indeed, we noticed multiple IMSIs in the selected signatures. In addition, Distalyzer identifies issues related to inefficient elements. However, it does detect incompatibilities where two elements are efficient while their combination has a high failure ratio.

By applying the same reasoning in Section VII-E3, one can find that Distalyzer has a linear computational complexity equal to $O(mkn)$ that can be approximated to $O(n)$. Therefore, Distalyzer has the same time complexity as the process of identifying major contributors in ARCD.

VIII. Conclusion

In this paper, we address the problem of automating the system that diagnoses cellular networks based on the data collected from large-scale monitoring systems, proposing a framework for root cause diagnosis in cellular networks. ARCD is unsupervised: it does not only automate expert analysis, but it carries it to a deeper level. Our tests, as well as the feedback from experts, show that we have promising results. In comparison to previous work, ARCD can run on a large number of features with categorial values, to identify the complex interplay between various features, and to provide an overview of the main identified malfunctioning devices and services, which can easily be double-checked by experts.

The three experts who have studied ARCD were extremely positive about the advantages it brings in their daily tasks. They highlighted the benefits of grouping elements related to the same problem into equivalence classes. They also appreciated the capacity of ARCD to identify occasional dependencies, and the time gain provided by the statistics-based hierarchical dependency discovering instead of manually updating the topology every time the network is modified.

The ARCD solution is also a starting point for a series of significant future studies. One of the first studies to conduct concerns parameter tuning and data labeling. Typically SDRs logs still require an expert to decide whether the logs report a successful experience. We aim to improve ARCD by reducing the number of parameters and finding a more efficient way to label data. We would also like to link our root cause diagnosis framework to an anomaly detection system within the same monitoring platform. This way the anomaly detector would trigger the root cause diagnosis process, ideally in real time, this being a key missing elements towards self healing cellular networks. Finally since some sessions matters more than other, we would like to explore new inefficiency definitions. This selection of open challenges and research questions paves the way toward fully autonomous self-healing mobile networks, having the capacity to diagnose the problem, to identify the root cause, and to fix the problem by reallocating resources or by updating software. We hope that reporting the choices that have made ARCD a successful network management product will inspire other researchers in the area.

References

[1] D. Palacios, I. de-la Bandera, A. Gómez-Andrades, L. Flores, and R. Barco, "Automatic feature selection technique for next generation self-organizing networks," *IEEE Communications Letters*, 2018.

[2] M. Mdini, G. Simon, A. Blanc, and J. Lecoeuvre, "ARCD: a solution for root cause diagnosis in mobile networks," in *CNSM*, 2018.

[3] E. J. Khatib, "Data analytics and knowledge discovery for root cause analysis in lte self-organizing networks," PhD dissertation, University of Malaga, 2017.

[4] A. Gómez-Andrades, P. M. Luengo, I. Serrano, and R. Barco, "Automatic root cause analysis for LTE networks based on unsupervised techniques," *IEEE Trans. Vehicular Technology*, 2016.

[5] I. Serrano Garcia, R. Barco Moreno, E. Jatib Khatiband, P. Munoz Luengo, and I. De La Bandera Cascales, "Fault Diagnosis in Networks," 2016, Patent Application WO2016169616A1.

[6] Z. Zheng, L. Yu, Z. Lan, and T. Jones, "3-dimensional root cause diagnosis via co-analysis," in *ICAC*, 2012.

[7] Y. Jin, N. G. Duffield, A. Gerber, P. Haffner, S. Sen, and Z. Zhang, "Nevermind, the problem is already fixed: proactively detecting and troubleshooting customer DSL problems," in *CoNEXT*, 2010.

[8] A. Mahimkar, Z. Ge, J. Wang, J. Yates, Y. Zhang, J. Emmons, B. Huntley, and M. Stockert, "Rapid detection of maintenance induced changes in service performance," in *CoNEXT*, 2011.

[9] C. Kane, "System and method for identifying problems on a network," 2015, US Patent 9,172,593.

[10] P. V. Klaine, M. A. Imran, O. Onireti, and R. D. Souza, "A Survey of Machine Learning Techniques Applied to Self-Organizing Cellular Networks," *IEEE Communications Surveys and Tutorials*, 2017.

[11] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, K. Papagiannaki, and P. Steenkiste, "Identifying the root cause of video streaming issues on mobile devices," in *CoNEXT*, 2015.

[12] J. Moysen and L. Giupponi, "A Reinforcement Learning Based Solution for Self-Healing in LTE Networks," in *Vehicular Technology Conference, VTC*, 2014.

[13] "3GPP; Technical Specification Group Services and System Aspects; Telecommunication management; Charging management; Charging Data Record (CDR) parameter description, TS 32.298, V15.5.1," Tech. Rep., 2018.

[14] N. Capit, G. D. Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounié, P. Neyron, and O. Richard, "A batch scheduler with high level components," in *CCGrid*, 2005.

[15] K. Nagaraj, C. E. Killian, and J. Neville, "Structured comparative analysis of systems logs to diagnose performance problems," in *NSDI*, 2012.

[16] R. Decker and F. Kroll, "Classification in marketing research by means of LEM2-generated rules," in *Advances in Data Analysis*, 2007.

Maha Mdini received her PhD from IMT Atlantique in September 2019 in data analysis applied to mobile network management. She worked at EXFO from 2015 to 2019 as a research engineer. She works currently as a postdoctoral researcher at RIKEN on weather forecasting using deep learning.

Gwendal Simon is a Full Professor at IMT Atlantique, an elite technological university in France. He graduated from University Rennes 1 with a PhD in 2004. He has been a researcher at Orange Labs from 2001 to 2006. Since 2006 he has been Associate Professor and then Full Professor at IMT Atlantique. He was a visiting researcher at University of Waterloo (Canada) in 2011-2012, and a visiting scientist at Adobe (US) in 2018-2019. His research interests include multimedia delivery systems and network management.

Alberto Blanc is an Associate Professor at IMT Atlantique since 2010. He received a ”laurea” in computer engineering in 1998 from the ”Politecnico di Torino,” Italy, and a Ph.D. in electrical engineering from the University of California, San Diego, U.S., in 2006. His research interests include performance evaluation of computer networks and cloud computing.

Julien Lecoeuvre graduated from ENSERG in 1993 – Grenoble INP (Ecole Nationale Supérieure d‘Electronique et de Radioélectricité de Grenoble) and specialized in telecommunications at McGill University in Canada. He worked for France Telecom Mobile and France Telecom R&D from 1994 to 2000, on cellular network deployment and performance optimization. In 2000, he co-founded Astellia (now part of Exfo), a world leader of Mobile Network Monitoring where he led the R&D for 11 years and then the Innovation department until 2018. Julien Lecoeuvre is now VP of Engineering at Acklio, specialized in solutions for IOT networks.