# Reflection: Automated Test Location Selection for Cellular Network Upgrades

Mubashir Adnan Qureshi*, Ajay Mahimkar†, Lili Qiu*, Zihui Ge†, Sarat Puthenpura†, Nabeel Mir†, Sanjeev Ahuja†
The University of Texas at Austin*, AT&T†

***Abstract*—Cellular networks are constantly evolving due to frequent changes in radio access and end user equipment technologies, dynamic applications and associated traffic mixes. Network upgrades should be performed with extreme caution since millions of users heavily depend on the cellular networks for a wide range of day to day tasks, including emergency and alert notifications. Before upgrading the entire network, it is important to conduct field evaluation of upgrades. Field evaluations are typically cumbersome and can be time consuming; however if done correctly they can help alleviate a lot of the deployment issues in terms of service quality degradation. The choice and number of field test locations have significant impacts on the time-to-market as well as confidence in how well various network upgrades will work out in the rest of the network. In this paper, we propose a novel approach – *Reflection* to automatically determine where to conduct the upgrade field tests in order to accurately identify important features that affect the upgrade. We demonstrate the effectiveness of *Reflection* using extensive evaluation based on real traces collected from a major US cellular network as well as synthetic traces.**

## I. Introduction

Cellular networks play a significant role in today's communication. To many users across different regions, it is often the only means of getting connectivity. Many users rely on cellular networks for entertainment, social activities and business critical tasks, such as stock trading, navigation, and emergency services. The cellular networks are extremely complex and constantly evolving at a rapid pace. Changes are introduced to either support new service features (*e.g.*, Voice over LTE), technologies (*e.g.*, LTE-advanced, small cells), fix software bugs, or expand network coverage. Deploying changes in a cellular network should be done with extreme caution in order to avoid any unexpected performance degradation or failures. Extensive testing is typically conducted in large-scale laboratory settings, but it is virtually impossible to replicate the large-scale, diverse variations and extreme complexity of real operational networks. Thus, the changes are tested on a smaller scale in the field. This small scale testing in the field is referred to as the *First Field Application (FFA)*.

The goal of FFA testing is to identify and infer the performance impacts of the change and make a recommendation for a go/no-go decision for a network-wide roll-out. If the desirable service performance impacts are observed after the FFA, the decision is to go-ahead with the roll-out. However, if performance degradations are observed, the changes need to be rolled back at the FFA locations and further analysis needs to be conducted in lab settings. Network operation and engineering teams carefully analyze the performance impact during FFA. Once they certify the change using field test results, the network-wide roll-out begins at a rapid pace. Strict deadlines are set to quickly update the network. Any unexpected issues discovered in the network-wide roll-out would slow down the process because of the need to understand the negative performance impact during FFA. Thus, careful planning and design of field tests is important to ensure smooth roll-out for the network-wide deployment.

A major challenge faced by the network operations and engineering teams in the planning and design of fields tests is *how to select network elements to be used for the field tests?* This is an important and unique challenge arising from the tremendous diversity in cellular networks. We present two illustrative examples to highlight this diversity. First, we analyze around 250 configuration parameters across 8000 LTE eNodeBs (*i.e.*, base stations in LTE) and observe that there are 747 unique clusters where each cluster is identified by a unique combination of configuration values. The cluster size distribution is not skewed implying diverse configuration settings across eNodeBs. Second, we note that for an example software upgrade case, different eNodeBs had different performance impacts; some had improvements after the upgrade whereas others had no impact. The root-cause for contrasting performance impact for the same software upgrade varies across different cases.

Current practice used by operations team is to test the change under high traffic, which assumes that negative impacts are more likely to occur under high traffic. Service performance indicators are also stable at high traffic locations, which enables accurate before/after analysis of upgrade impacts. However, contrasting performance impacts can happen often due to features unrelated to traffic. Also past upgrade history data cannot always be used to guide the design of FFA tests because the new upgrades may introduce features/configurations changes which were never deployed in the past.

**Our approach:** We propose a new approach *Reflection* that systematically utilizes a diverse set of features (*e.g.*, software/hardware configuration, radio parameters, user population, mobility patterns) and automatically identifies the FFA locations that would improve the predictability between the performance impacts during FFA and network-wide deployment. Having predictable performance behaviors with

978-1-5090-6501-1/17/$31.00 

FFA ensures a smooth and rapid wide-scale roll-out. Though we apply *Reflection* in cellular context, the techniques we develop are generic enough to be applied in other domains by appropriately selecting the set of features.

Designing *Reflection* requires us to address the following technical challenges: (i) **Extremely large search space:** There are tens of thousands of cellular eNodeBs to choose from, each with hundreds of features. Which features to consider and which nodes to test have significant impact on the accuracy and predictability of the tests. Given $N$ features and each can take $k$ values, it generate $k^N$ test cases. For example, $N = 30$ and $k = 2$ (binary features) generates around 1G test cases, which is already infeasible for operational networks. (ii) **Interactions between features:** It is often impossible to know in advance which features will interact negatively with a new network change. Particularly, for a new feature/configuration introduction, you have no prior upgrade history data and there is uncertainty in knowing how this new feature will interact with the rest of the system. Ideally we should automatically discover any undesirable interaction based on the limited FFA tests, and resolve the issue before the network-wide roll-out. (iii) **Very low sampling for FFA locations:** Since one of the goals of FFA is to minimize the risk of negative impact on network locations, the network operations and engineering teams have a very low sampling budget. For example, for 10,000 eNodeBs, the number of locations available for FFA testing could only be 100, yielding a sampling rate of $1\%$. Such a low sampling rate and wide-variety of features pose a significant challenge to identify the appropriate set of locations for FFA with high predictability during network-wide roll-out.

One way to design test cases is to diversify all features (*e.g.*, for each feature, select test case that involves different values of that feature). However, the number of test cases grows exponentially with the number of features, which can be prohibitively expensive. In practice, only a small number of features are important to the performance, and these important features may not be known in advance.

There are several works in the area of fractional factorial design and software testing, where a small subset of experiments are chosen out of exponential number of possible experiments and the hypothesis is that testing on this small but carefully chosen subset will expose important features. However, these techniques are not directly applicable in our case because: (i) Some of the experiments in the subset outputted by software testing/fractional factorial design do not exist in the real data, and (ii) the outcome of an experiment in our case is probabilistic whereas the outcome is assumed to be deterministic in software testing or fractional factorial design based approaches. Therefore, we develop a *multi-phase test planning*. In the first phase, we iteratively pick nodes that offer the best coverage over all features so that we can assess the impact of each of these features to narrow down to a smaller subset of candidate features that are likely to be important. In the later phases, we focus on testing only these candidate features by selecting nodes that offer the best coverage over the candidate features so that we can further narrow down to the final set of features that we consider as important. In this way, we can significantly reduce the number of test cases. Our contributions can be summarized as follows:

1. We develop a multi-phase test planning in *Reflection* to automatically diagnose contrasting performance and select network locations for FFA testing to improve the predictability for network-wide deployment (Section II).
2. We develop an algorithm to analyze contrasting performance impact across network locations during the FFA. It helps narrow down the relevant features to be considered for the next phase of planning. In comparison, the previous works (*e.g.*, Litmus [19] and Mercury [20]), can only detect contrasting performance impacts but not identify the root cause (Section II).
3. We evaluate *Reflection* using one-year data collected from a major cellular service provider in US. Our results show that our diagnosis approach accurately identify the root causes and our test planning only requires testing 1-1.5% nodes to identify the major features that affect degradation. We further evaluate Reflection using synthetic traces.

## II. *Reflection* DESIGN

We develop a multi-phase test planning approach that uses the performance outcome from previous tests to guide the design of the subsequent tests. The intuition behind our multi-phase test planning approach is that if we know the performance at eNodeBs with different feature values, we can more effectively design the tests. This multi-phase test planning is practical since major cellular service providers schedule FFA in a staggered manner. Staggered roll-out is necessary because thousands of eNodeBs cannot be upgraded on a single day and rolling out the upgrade over multiple days also enables the operation teams to carefully monitor their performance impacts. Thus future tests can be designed using performance impact assessments from the previous tests.

In order to realize the multi-phase test plan, we need to answer the following important questions: (i) What features to use for test planning and performance analysis? (ii) How to prepare inputs for test planning and diagnosing contrasting performance impacts? (iii) How to determine the initial test locations? (iv) How can we determine the performance impacts? (v) How to diagnose the contrasting performance in the previous test? (vi) How to use the analysis results of the previous tests to design future tests?

To answer these questions, we design *Reflection*, shown in Figure 1. It first extracts the features to be used for test planning and then processes them for analysis. Then it performs the first phase test planning (Section II-C), applies upgrades to the testing locations, assesses performance impact (Section II-D), diagnoses test performance to identify important features (Section II-E), uses the diagnosis results to guide subsequent test planning (Section II-F), and iterates until the maximum number of phases is reached. In the end, it outputs the final diagnosis results for the important features and predicts the performance on untested locations. Sections

II-A and II-B are specific to our cellular networks, while the other parts are general and can be applied in other contexts.

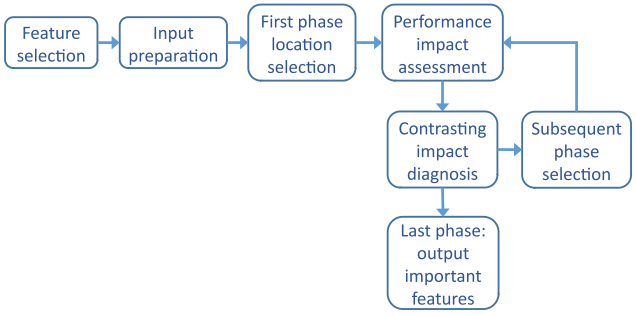



Fig. 1: Reflection workflow.

### A. Feature Extraction

Our traces come from a major LTE service provider in the US. E-UTRAN NodeB (abbreviated as eNodeB) in LTE is essentially a cellular eNodeB. It communicates directly with mobile handsets (UEs). It runs the E-UTRA protocol, which uses OFDMA for the downlink and single-carrier FDMA (SC-FDMA) for the uplink. In order to make the selection effective, it is important to include all features that can potentially affect eNodeB performance. Therefore, we try to be comprehensive and obtain as much information about eNodeBs as possible.

We come up with a few hundred features for each eNodeB by digging through configuration files and collecting measurement driven data. We group the features into the following categories: (i) node-level configuration(*software version, hardware version, modem, eNodeB manufacturer etc.*), (ii) protocol-level configuration(*Carrier-Aggregation enabled, Carrier frequencies etc.*), (iii) topological features(*connected MME ID, load-balancing enabled etc.*), and (iv) location centric features( *traffic, mobility, terrain type etc.*). The set of features used here are more or less the ones used by operations in diagnosing the performance issues. These features are particular to cellular domain; *Reflection* can easily be applied in other contexts by selecting the relevant features.

### B. Input Preparation

The input to our planning and diagnosis algorithms consists of eNodeBs' feature list and its performance tag, which indicates whether its performance improves, has no change, or degrades. We process these raw inputs as follows.

**Discretize feature values:** Most features in our traces are binary. The remaining features take either textual values or real numbers. To ease diagnosis and test planning, we systematically map all textual features to numerical values. In addition, we discretize a real numbered feature by comparing it with $mean - 2 * std$ and $mean + 2 * std$ to map to one of the three levels: 0, 1, 2, where both the mean and standard deviation ($std$) are computed using values across all eNodeBs. We then convert them to binary features.

**Dimension reduction:** We propose clustering features into equivalence classes. This has several benefits. First, it is impossible to differentiate the impact of two features if (i) they always change together and (ii) for each value of feature $f1$, there is a unique value for feature $f2$. Consider two features $f1$ and $f2$. When they take 00, performance improves. When they take 11, performance degrades. The traces do not have instances with the feature values of 01 or 10. In this case, it is impossible to tell if performance degraded due to $f1 = 1$ or $f2 = 1$ or ($f1 = 1$ and $f2 = 1$). Second, clustering features reduces the number of unknowns and improves accuracy.

To accommodate such inherent ambiguity as well as improve accuracy and speed, we group features into equivalence classes. Strictly speaking, two features are indistinguishable whenever there is always one unique value of $f2$ for each value of $f1$ and vice versa. In practice, we relax this condition to allow occasional violations as long as in most cases there is one unique value of $f2$ for each value of $f1$ and vice versa.

For a pair of features $f_i$ and $f_j$, we compute the following metric to determine if the two features are equivalent. For each value $f_i$ takes, say $v_{i,k}$, we look at how many unique values $f_j$ takes and the number of nodes that take these values to compute a metric, called *unique ratio*: $(\frac{\sum_k max_l N(v_{i,k},v_{j,l})}{\sum_k \sum_l N(v_{i,k},v_{j,l})} + \frac{\sum_l max_k N(v_{j,l},v_{i,k})}{\sum_l \sum_k N(v_{j,l},v_{i,k})})/2$. where $N(v_{i,k}, v_{j,l})$ is the number of nodes that take the $k$-th value in feature $i$ and take the $l$-th value in feature $j$. We search for the most popular value $v_{j,l}$ that feature $j$ takes when feature $i$ takes the $k$-th value. $max_l N(v_{i,k}, v_{j,l})$ is the number of nodes whose feature $j$ takes the most popular value $l$ under $v_{i,k}$. Collectively, the numerator in the first term of the above equation reflects the total number of nodes taking the most popular feature values $v_{j,l}$ normalized by the total number of nodes from the perspective of feature $i$, and the second term computes the same quantity from the perspective of feature $j$. We normalize by 2 to get the mean as the equivalence relationship is symmetric. When the unique ratio is higher than a threshold, we declare that two features are equivalent. The threshold must be high enough so that we group only those features that are almost identical. We set it to 0.95.

**Upgrade trigger:** In our measurements sometimes all nodes have the same value for a feature, and then all change to another value for the same feature upon application of update. Among these nodes, some see improvement while others see degradation. At the first glance, one may think this feature is irrelevant, since nodes with the same value in the feature see different performance. But in practice, this feature could be relevant and the degradation could be due to interaction between this feature and some other features.

To handle such cases, features that changed during upgrade are considered as possible triggers to the performance issues. For the features that take different values across different eNodeBs at a given time, we can rely on our diagnosis algorithm(Section II-E) to identify them. So the trigger set consists of the set of features that changed from one value to another when upgrade gets applied. Then we apply the diagnosis algorithm described in Section II-E to identify root causes. Our diagnosis result includes the trigger set and root cause,

where the trigger set contains features that changed during the upgrade and the root cause contains the equivalence classes of features that best explain the contrasting performance.

### C. First Phase Test Location Selection

In this subsection, we focus on design of the first phase. Its main goal is to diversify over all features to identify a small set of features that are likely to matter. This step can quickly prune irrelevant features to get a small set of candidate features. We diversify over all the features because for the new upgrade, we may have no knowledge about which features are going to matter. If that information is known, then diversification can be enabled only on the selected features. The subsequent phases focus on this small set of features to pinpoint the root cause. We use 70% budget in first phase because we want to enable diversification over a large set of features and keep 30% for subsequent phases where number of features may be small.

**Hamming distance based selection:** We develop a greedy algorithm that selects feature values that maximize the minimum Hamming distance among them. Minimum Hamming distance is used as the optimization metric because it captures how many features whose impacts can be assessed (*e.g.*, if we select two feature values: 000 vs. 111, we can potentially assess the impact of three features by comparing the performance when each feature takes a value 0 versus 1). In the next iteration, we add a feature value combination that has the largest Hamming distance from the one selected earlier. For example, if we first select a feature combination 00. Then we will prefer to select a feature combination 11 instead of 01 or 10. This is because it allows us to potentially estimate impact of two features by computing the difference between when one feature takes a value 0 versus takes a value 1. In the third iteration, we pick a feature combination that maximizes the minimum Hamming distance from the two we picked so far: $max\ min_i\ Hamming(n_i, n')$, where $n_i$ are the set of feature combinations already selected and $n'$ denotes the new feature combination to add. We iterate until we select enough eNodeBs to test. In order to compute degradation probability, we need to select multiple nodes from each feature combination. Our evaluation uses 3 nodes per feature combination. We also tried 2 and 4 but that did not change the performance much. We want to test more clusters so that we have feature diversity in our test cases. Sampling a large number of nodes from each cluster reduces the number of clusters that can be sampled. For *k* features, *n* unique feature combinations and *m* selected feature combinations, the running time is $O(nm^2k)$.

We can support a number of other extensions in the same framework. For example, one can also weigh the Hamming distance by the importance of a feature. The weight can reflect either the frequency of occurrence or the importance of a given feature based on prior knowledge. We plan to explore these extensions as part of our future work. While our Hamming distance based approach appears simple, it provides good empirical performance. There may be interesting connection between Hamming distance and sub-modular optimization (e.g., [3]). This paves the way towards using the machinery of sub-modular optimization to come up with efficient greedy algorithms to choose the best subset of base-stations [22].

### D. Performance Impact Assessment

The network operation teams carefully monitor the impacts of network changes using a wide variety of service performance indicators. If there is performance degradation after the network upgrade, a roll-back to the previous configuration is important to minimize the service disruption. Statistical techniques such as Mercury [20] and Prism [18] provide automated ways to detect the impact. We use Mercury to automatically extract the performance indicator for each eNodeB about whether its performance degrades after an upgrade or not. We use the following service performance metrics in Mercury to capture the statistical changes in behaviors: (i) *Accessibility* is the ratio of successful call establishments to total call attempts, (ii) *Retainability* is the ratio of successful call terminations to total calls, and (iii) *Data throughput* is a measure of bits/sec delivered to the end-users. Unless otherwise specified, we conclude a node degrades if any of the above three metrics satisfies the following condition: $\frac{P_{before} - P_{after}}{MAD} > threshold$, where $P_{before}$ and $P_{after}$ denote the median performance during the 14 days before and after the upgrade, respectively, $MAD$ stands for mean absolute deviation during the 14 days before the upgrade, which is defined as $\frac{1}{n}\sum_i \|x_i - mean(x)\|$, and $threshold = 3$. We use 14 days before and after the application of upgrade to analyze performance impact because this is the current operational standard in Mercury.

### E. Contrasting Impact Diagnosis

Once we obtain the results from the first phase of testing eNodeBs, we can use the eNodeBs that see contrasting performance to identify the important features that affect network upgrades. Specifically, if there are contrasting impacts for the same type of upgrade but across different network locations, we identify the root-cause that best explains the contrast.

**Problem Formulation.** Each eNodeB can be characterized by $N$ features. Because degradation is a probabilistic event so even when two nodes take identical values in all features, one may degrade while the other may observe no performance change. So we use degradation probabilities instead of a binary performance tag. For each unique feature value combination, we compute degradation probability based on our traces. For example, when there are two binary features $f_1$ and $f_2$, we compute the degradation probabilities when they take 00, 01, 10, and 11, respectively. Then the goal is to identify which subset of features can best separate the high degradation probabilities from the low degradation probabilities.

The major questions we should answer in diagnosing upgrade performance issues involve: (i) what metric can best capture the notion of separation between degradation probabilities, and (ii) how to design an efficient algorithm that can handle a large number of features, which can be a few hundred in our traces, and it is too expensive to try all possible combinations.

We summarize the limitations of existing approaches here and present diagnosis results over real case studies in Section III-B.

**Existing approaches and limitations.** *chi-squared test* is used to determine if two events are independent. One way to apply chi-squared test is to test the dependence between the degradation probabilities and a given feature, and select the most dependent features. *Information gain* is used in building decision trees and it measures the importance of an attribute using entropy. *Fisher score* finds a subset of features such that in the data space spanned by the selected features, the distance between data points in different classes are as large as possible while the distance between data points in the same class are as small as possible. *Linear regression* can also be applied to diagnosis. We form a matrix $A$ based on the unique feature values and form a vector $b$ based on the corresponding degradation probabilities. To learn the importance of each feature, we construct a linear equation: $Ax = b$, where $x_i$ is the weight of the $i$-th feature. We can simply solve $x$ based on the linear equation. Often we may not have enough observations to uniquely solve $x$. To address the under-constrained problem, one can use regularization terms. Ridge regression incorporates $L_2$ norm regularization and lasso regression incorporates $L_1$ norm regularization.

Our evaluation indicates that the accuracy of these existing algorithms is limited especially when the root cause contains multiple important features. A closer look of the results reveals several significant limitations. First, they rank the features based on a certain metric, and pick the top ranked few features. But there can be significant correlation among these features. Therefore, we should revise the algorithm to make them iterative and remove the impact of the previously selected features before picking the next important feature. Second, the existing metrics fall short. For example, the chi squared test fails to take into account different sample sizes in different feature values. It performs poorly when one of the feature values is more dominant in population than the other. Both information gain and fisher scores are biased towards a feature that has more diverse values. For example, suppose most features take two values and one feature takes 10 values. The feature with 10 values tends to be picked as the root cause since its information gain and fisher score tend to be higher. Linear regression accuracy is also limited due to (i) dependence between the features, (ii) non-linear relationship between the features and degradation probabilities, and (iii) significant under-constrained systems.

**Our Approach.** We use an iterative algorithm to analyze the contrasting impact. For each unique feature value combination, we compute the degradation probability using the traces. Our approach iteratively adds one feature set at a time to optimize a metric. A feature set can consist of 1-3 features. We use 3 as the upper bound because the operations team observes root cause usually consists of 1-2 features(Section III-B). Following the standard $Z$-test [24], we develop a new metric:

$$\frac{\sum_i \sum_{j \neq i} |z_{ij}|}{\|\#regions\| \times (\|\#regions\| - 1)}$$

where $z_{ij} = \frac{p_i - p_j}{\sqrt{p_{ij}(1-p_{ij})(1/n_i + 1/n_j)}}$ and $p_{ij} = \frac{x_i + x_j}{n_i + n_j}$, $i$ and $j$ denote one of the feature combinations defined by the currently selected features (*e.g.*, 00, 01, 10, 11 for two binary features), $\|\#regions\|$ is the total number of regions defined by the selected features (*e.g.*, two binary features define 4 regions: 00, 01, 10, 11), $x_i$ and $x_j$ are the number of degraded eNodeBs for the $i$-th and $j$-th feature combinations, and $n_i$ and $n_j$ are the corresponding total number of eNodeBs.

Our metric reflects how different are the two distributions. Not only does it capture the difference between feature values but also difference between population sizes. Essentially, it quantifies the average difference between $z$-scores across all regions defined by the selected features. The intuition is that important features yield larger difference in degradation probabilities. But instead of directly using probability difference, we weigh the probability difference based on the number of samples in the cluster since a large difference under a small sample size does not mean much but the same difference under a large sample size means more.

Applying the greedy algorithm, we first add the feature set $f_{k1}$ that maximizes our metric. Then we add a second feature set $f_{k2}$ that yields the maximum difference when these two features take different values. For example, for binary features, we want to maximize performance difference when $f_{k1}$ and $f_{k2}$ take 00, 01, 10 and 11. We iterate until the difference across different regions does not increase significantly.

### F. Subsequent Phase

After we analyze the performance results and narrow down the potentially important features, our subsequent phase in test planning refines selection. We can have two or more phases. Note that the total budget does not change, it is just split into different phases. The second and all subsequent phases essentially use the same procedure. So we focus on the second phase. It is similar to the first phase: it also employs a similar greedy algorithm that maximizes the minimum Hamming distance between selected nodes. There are two main differences between the first and second phases. First, since the first phase already narrows down to a subset of candidate features, the second phase only diversifies over these candidate features. Second, the second phase should add new nodes to test that complement the nodes already tested during the first phase. This can be achieved by selecting a new feature value that maximizes the minimum Hamming distance from all the selected nodes so far, including those selected in the first phase and the previous iterations of the second phase.

After testing on these eNodeBs selected during the second phase, we run the same diagnosis algorithm in Section II-E. Note that by now we see the performance outcomes from all nodes selected in all test phases, so we use the performance information from all tests to identify the important features that contribute to performance difference. The only difference is that the intermediate diagnosis steps use a lower improvement threshold to pick more features and avoid missing important features for designing future tests whereas the final diagnosis step uses a higher threshold to limit the false positive. Our

evaluation uses 0.005 during the intermediate diagnosis and 0.03 during the final diagnosis.

## III. Evaluation Using Real Traces

In this section, we present an overview of the case studies that we will use for evaluation of our diagnosis, test planning and prediction. Each of the case study involved exhaustive analysis by the field operation teams of a large tier-1 cellular service provider in understanding the performance impacts of network changes during the large-scale roll-out. The number of case studies where the contrasting performance impact was seen at a large scale are limited. We separately evaluate diagnosis for small scale case studies in Section III-B2. In our evaluation, we will show that our automated diagnosis algorithm would have enabled accurate and rapid identification of the root-cause upon seeing contrasting performance impacts.

### A. Evaluation Methodology

We use the service performance data and apply Mercury [20] algorithm to conduct a statistical pre/post impact analysis around network changes to mark each eNodeBs with Degradation or No Change, by comparing their performance before and after the upgrade. The time-interval for pre/post analysis is -14/+14 days. For each eNodeB, approximately 250 configuration and metric-driven features were collected. These features comprised both textual as well as numerical values, and we systematically convert these values into a few discrete numbers as described in Section II-B. Each case study has a different number of eNodeBs. We evaluate diagnosis and test planning components separately and report the average over 25 random runs for each scenario.

**Evaluating diagnosis accuracy:** We first evaluate the diagnosis step using our metric and compute accuracy by comparing with the ground truth root causes from operations team.

**Evaluating test planning:** Next we evaluate the effectiveness of test planning. Test planning evaluation differs from diagnosis evaluation in that diagnosis sees the performance outcome from all eNodeBs involved in the upgrade. Whereas to reduce test cost test planning only allows to test upgrade on a smaller subset of nodes and the input to the test planning contains the performance from these tested nodes along with their features. The accuracy in test planning depends on both selection of test locations and diagnosis algorithm. We evaluate our approach against *Random* and *High-Traffic* based selection approaches. In high-traffic based selection, we select nodes from a group of nodes for which the traffic is in $85^{th}$ percentile or above.

We use the following metrics to quantify the accuracy: $recall = \frac{tp}{tp+fn}$ and $precision = \frac{tp}{tp+fp}$. where $tp$ is the number of true positives (*i.e.*, correctly detected important features), $fp$ is the number of false positives (*i.e.*, incorrectly detected important features), $fn$ is the number of false negatives (*i.e.*, missed important features). We also integrate precision and recall into a single metric called $F1\ score$, which is the harmonic mean of precision and recall: $F1\ score = \frac{2}{1/precision+1/recall}$. For all three metrics, larger values indicate higher accuracy and maximum value can be 1.

We first quantify the recall of the first test phase based on whether the ground truth features are included in the analysis. We are only interested in recall in the first step since the second step can rule out false positives. Next we evaluate the accuracy of our second step test phase using F1 score. The average F1 score of of the multi-phase approach is 35% higher than the single-phase, and the detailed result of the single-phase is omitted in the interest of space.

**Evaluating prediction:** Finally, we quantify the accuracy of predicting whether an eNodeB will degrade based on its feature value and the performance outcome of a few tested nodes. The prediction accuracy is measured as $\sum_i |D_i^{real} - D_i^{est}|$, where $D_i^{real}$ and $D_i^{est}$ are the actual and estimated number of degraded eNodeBs that take feature value $i$, respectively. We normalize it with the number of nodes in the case study. The prediction error is a little high in the traces because (i) degradation is a probabilistic event and the probability estimation can be erroneous especially without a sufficient number of samples, and (ii) a heuristic is used to tag eNodeBs with degradation in the ground truth data, which may introduce additional error. Our primary objective is to identify the root cause for contrasting impacts during FFA. While we only have five real traces where we know the exact ground truth, we have access to more traces that contain eNodeBs performance before and after upgrade. We evaluate prediction error for these additional traces in Section III-D.

### B. Diagnosis Evaluation

We collect ground truth information from the operation teams of the cellular service provider. We divide it into two categories: (i) diagnosing performance impact of *network-wide* changes, where the same change is applied to multiple eNodeBs across the network, and (ii) diagnosing performance degradation of *individual eNodeBs*, where only one node may experience the change and we apply our algorithm to identify the root cause by comparing it to neighboring eNodeBs that did not experience the degradation around the same time. Category (i) consists of 5 case studies of network-wide changes (Section III-B1) and category (ii) consists of 261 case studies over two months (Section III-B2).

*1) Diagnosing Impact of Network Upgrades:* **Case Study I: Software upgrade on LTE cell towers.** In our first study, we analyzed the roll-out and impact analysis of a software version upgrade across multiple LTE cell towers. We observed performance improvements on some cell towers and minor performance degradation (increase in call drops) on a small set. The operation teams figured out that the end-users were sending duplicate statistics measurements to the cell towers that were pointing to the same reference object. This led to cell towers dropping the calls because of internal failures. This effect was only observed on cell towers with the specific configuration; however other cell towers with the same software upgrade but different configuration did not experience service performance degradation. We run our greedy diagnosis on the input that contains 250 features for

each eNodeB and whether its performance degrade or not. It identified UTRAANR (UMTS Automated Neighbor Relations Discovery) feature as the possible trigger and Duplicate Measurement as the root cause. Both the trigger and root cause match exactly with the ground-truth from the operation team. Our scheme further estimates degradation probabilities are 0.67 and 0.28 when Duplicate Measurement is present and not present, respectively.

**Case Study II: Configuration changes on LTE cell towers.** In our second case study, one of the configuration changes introduced a new positioning feature for the end-users. The positioning feature enables users to report the time difference between specific signals from several cell towers to serving centers in the core network. This feature introduction led to increase in call drops only at cell towers with a higher number of handovers. Again, the diagnosis of the contrasting impact was challenging. Our automated algorithm enabled faster detection of the root-cause. It identified Observed Time Difference of Arrival (OTDOA) [6] Hearability Enhancement (time synchronization for UE location positioning) as the possible trigger and handovers as the root case. Both match with the ground-truth from the operations team. It further derives the degradation probabilities of 0.65 and 0.42 when the number of handovers is high and low, respectively.

**Case Study III: Hardware updates in the core.** We started with hardware update in the core network at Mobility Management Entity (MME). MME in LTE manages multiple cell towers and is responsible for processing the signaling information between end-user and core network. After the hardware change, we observed that there was an increase in a particular type of alarm across a small number of cell towers but not everywhere. Our diagnosis discovered that the software version on the cell towers was the cause. A specific software version had conflicting interactions with the new hardware controller in the MME and caused the increase in the number of alarms. Our algorithm identified controller type as the trigger, and OS version as the root cause for raising alarms on MMEs, which agrees with the ground truth from the operation teams. It further derives the degradation probability of 0.83 in OS version 1 and 0.55 in OS version 2.

**Case Study IV: Software upgrade on LTE cell towers.** The fourth case study came to us before the operation team knew the ground truth. We applied our algorithm to understand the contrasting service performance impacts resulting from a software roll-out on LTE cell towers in a specific region. There was an increase in connection establishment failure rate at only a small number of cell towers. Our algorithm automatically identified that congested towers had performance degradation, whereas others had no negative impacts. Congestion on the cell towers was because of a multi-day high traffic special event scenario which coincided with the day of the software upgrade. Our results helped the operation team, who later confirmed that issue occurred due to high traffic during holidays.

**Case Study V: Software upgrade on LTE cell towers.** In our final case study, we analyzed software upgrade that was being rolled out on LTE cell towers across the entire network. The operation teams had noticed contrasting performance impacts across cell towers. We used Mercury to detect that some cell towers experienced throughput degradation whereas other cell towers had no negative impact. We automatically identified the cell towers that were serving a large number of users and carrying higher traffic were experiencing degradation in LTE data throughput. We confirmed our findings with the operation teams. It turned out that the new software version was unable to handle high traffic on specific carrier frequencies.

Table 1 compares the diagnosis results under different diagnosis algorithms for five case studies. All algorithms except ours miss some case studies.

| Reflection | Info. Gain | Fischer Sc. | L1/L2 Norm | Chi Square |
|---|---|---|---|---|
| 100% | 60% | 80% | 40%/80% | 80% |

TABLE I: Diagnosis accuracy across five case studies.

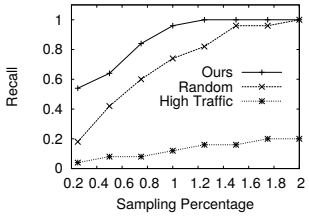
*2) Diagnosing degradations on individual eNodeBs:* We analyzed the performance degradations observed on individual eNodeBs during a time interval of two months. For each case, our objective is to sift through a wide variety of feature data sets including configuration, alarms, performance and traffic metrics, and identify the potential explanation for the degradation. Degradations were observed in service performance metrics, such as accessibility, retainability, and data throughput. The operation teams had manually gleaned through the feature sets to identify the root-cause. Even though the traces are smaller-scale compared to the previous ones, it still takes at least a few hours to manually diagnose each case. In comparison, we develop a tool that automates the diagnosis completely: it pulls the database to get the raw input, prepares the input for our diagnosis program, and automatically diagnoses each case within 2 minutes.

The number of features for each case study ranged from 60 to 80. For each eNodeB with performance degradation, we identified a number of neighbors that did not experience degradation around the same time. This gives us a contrasting set. Since the cases were tackled historically, we knew the ground truth root-cause for each of them. Our tool identified radio congestion (significant increases in random access channel request) as the most likely explanation for the performance degradation. Other root causes include remote IP address unreachable, cell range change after software upgrade. This aligns perfectly with the findings from the operation teams.

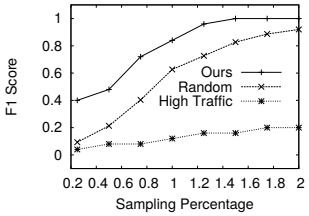
### C. Test Planning

**Case Study I:** Figure 2(a) plots the recall after the first step as the sampling ratio is varied from 0.25% to 2%. We make several observations. First, Hamming distance based node selection yields significantly higher accuracy than random selection and high traffic selection. High traffic selection under-performs even the random selection because it only sees a biased subset of nodes. The bias is particularly bad in this case because the feature value that causes degradation is more popular in low traffic locations and more low traffic nodes see performance degradation. This highlights the importance

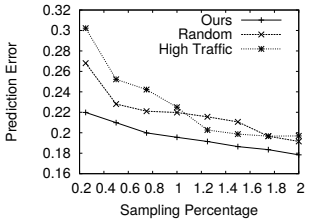
of sampling diverse feature values in the tested nodes instead of pre-assume traffic is the feature that matters. Second, the performance benefit of strategic node selection is largest when the sample size is small. Due to a large number of eNodeBs in the network, a sample size of 0.5% - 1.5% is desirable according to the operation team, and we observe a large benefit in those scenarios. Third, it is interesting to note that just by sampling 1%, Hamming distance based selection combined with greedy diagnosis yields close to 100% recall. This significantly reduces testing cost and manual diagnosis effort. In comparison, the other selection schemes yield only around 10%-70% around this region. Figure 2(b) compares F1 score of the second step. Hamming distance based continues to out-perform the other two schemes. The gap between random and Hamming distance reduces due to more samples are drawn in the second step. Figure 2(c) shows the prediction errors of different schemes. Prediction error for *Reflection* approaches 0.18 and for Random and high-traffic based heuristics, it approaches 0.2 as sampling ratio approaches 2%.




(a) Step 1 recall




(b) Step 2 F1 score



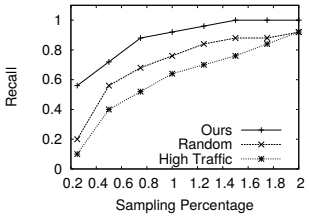

(c) Step 2 prediction error

Fig. 2: Test planning in case study I.

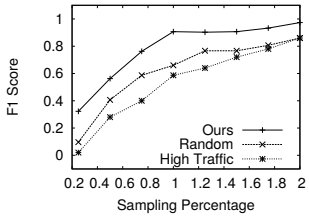
**Case Study II:** Hamming distance based selection continues to perform the best, while random selection and high traffic based selection perform significantly worse especially when the number of samples are small. High traffic based selection performs the worst because high handover nodes are distributed proportionally across low-traffic versus high-traffic nodes. So high traffic based selection is essentially randomly sampling over a smaller pool of nodes.

Figure 3(b) and (c) plot F1 score and prediction error after the second step using the traces from the case study II. Hamming distance based selection yields the highest F1 score and lowest prediction error. Sampling 1% nodes yields F1 score of 0.9 and prediction error of 0.18. The F1 score approaches to 1 and prediction error approaches 0.14 as the sampling ratio increases to 2%.

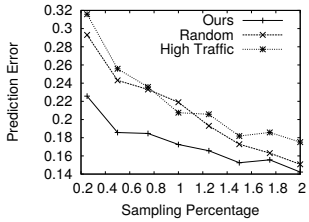
**Case Study III:** Figure 4(a), (b), and (c) plots recall after step 1, F1 score and prediction error after step 2, respectively. In this case, *Reflection* performs significantly better than the




(a) Step 1 recall




(b) Step 2 F1 score

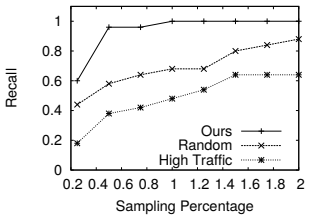



(c) Step 2 prediction error

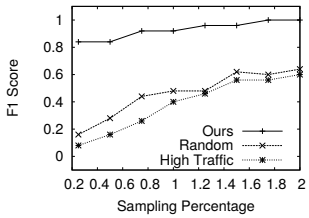
Fig. 3: Test planning in case study II.

other schemes due to non-uniform feature value distribution of the root cause feature. Other schemes tend to pick very few nodes which do not have popular feature values. Due to the absence of diverse values of root cause feature in testing set, these schemes fail to capture the root cause.

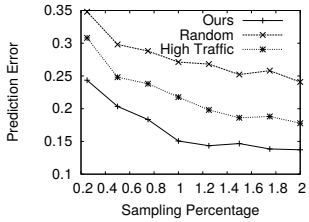
We do not evaluate test planning in case studies IV and V because these cases have within 100 nodes and test planning is useful for a large number of nodes.




(a) Step 1 recall




(b) Step 2 F1 score

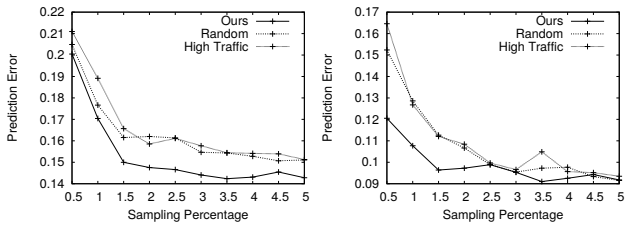



(c) Step 2 prediction error

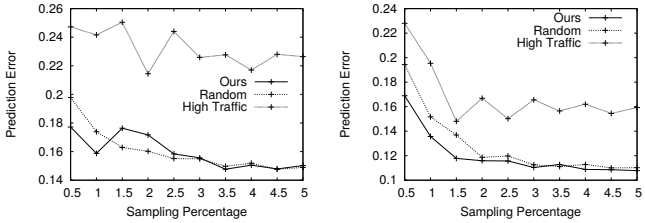
Fig. 4: Test planning in case study III.

### *D. Prediction Error*

In addition to the prediction error results for the case studies, we also evaluate prediction error using a few more traces. Figure 5 summarizes the results. As we can see, Hamming distance based selection yields the lowest error. Sampling 2% nodes can already achieve prediction error of 0.1-0.18. High-traffic based selection performs either similarly to random selection or worse since traffic is likely not the root cause for degradation in these traces; picking high-traffic only nodes causes unnecessary bias in sampling and lead to degradation in traces 8 and 9.




(a) Trace 6 (b) Trace 7




(c) Trace 8 (d) Trace 9

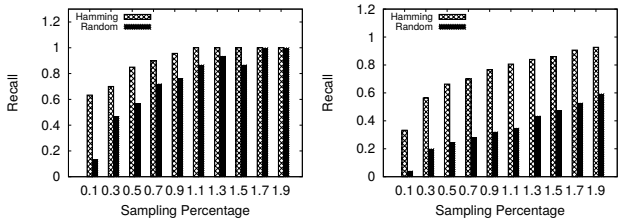
Fig. 5: Prediction error in additional traces.

## IV. Evaluation Using Synthetic Traces

In this section, we complement the real trace analysis by generating synthetic traces. While synthetic traces are not realistic, they allow us to extensively evaluate each component in our design and provide insights on how each parameter impacts the accuracy of diagnosis and test planning. We randomly select $K$ important features out of $N$ total features. Then we generate a degradation probability for each combination of $K$ important features. For each configuration, we generate a new trace with different eNodeB degradation probabilities and feature values.

### A. Test Planning

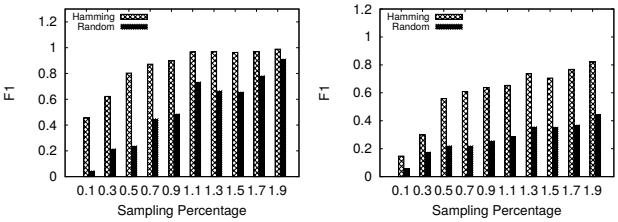
For each configuration, we generate 25 random traces, vary the sampling ratio and report the average. We generate 20000 eNodeBs each with 50 features. We compare our Hamming based selection with random selection for the two-phase planning. We plot comparison graphs for two cases, when we have 1 important feature and when we have 5 important features. Figure 6 shows the recall of Hamming based approach is higher because we diversify the feature values at each step. The benefit of Hamming selection increases as the sampling rate decreases and/or the number of important features increases since the test planning algorithm is more critical in these cases.




(a) 1 important feature (b) 5 important features

Fig. 6: Recall after step 1 in synthetic traces.

Figure 6 plots the recall after the first phase and Figure 7 plots F1-score after the second step. For 1 important feature around 1% sampling rate, recall and F1 is close to 1 for *Reflection* while the random selection clearly underperforms. The

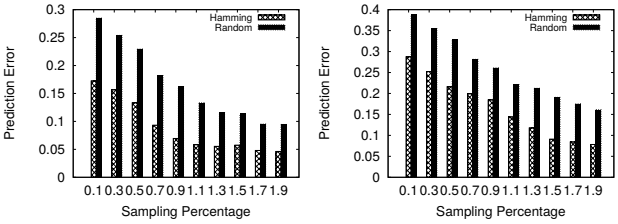



(a) 1 important feature (b) 5 important features

Fig. 7: F1 scores after step 2 in synthetic traces.

difference in F1 score is around 0.4 at 0.9% sampling ratio. For 5 important features around 1% sampling rate, *Reflection* can identify 4 out of 5 important features whereas random can only identify at most 2 important features. Figure 8 shows the prediction error after the second step where the difference between Reflection and random can be as large as 15%.

The advantage of Hamming distance based selection over random increases when the feature value distribution is skewed in the dataset. In those cases, random selection misses out on picking certain feature values for testing so it becomes difficult to identify the root cause for contrasting performance impacts.




(a) 1 important feature (b) 5 important features

Fig. 8: Prediction error in synthetic traces.

## V. Related Work

**Software testing:** Software testing includes white-box testing using source code, black-box testing without source code, and grey-box testing based on specification of the program. Combinatorial testing is used to generate automatic test cases using various features of the software (*e.g.*, [23], [8]). Software testing differs from our upgrade tests in several important aspects as we explained in Section I. A/B testing [14], [13] are used in the web domains for making data-driven decisions. However, it is prohibitively expensive to perform A/B testing on all possible feature combinations. [5] develops an algebraic approach to determine which network paths to probe in order to fully recover end-to-end path properties. The number of paths that it requires to monitor is close to the number of links in the network. In comparison, we can only sample a very small fraction of eNodeBs.

**Network upgrades and impact analysis:** [11] proposes a fast system for executing consistent updates in a software defined network. [25] proposes abstractions for network updates and preserves consistency between configuration transitions. [9], [16], [7] achieve congestion-free upgrades with zero packet loss. Magus [33] minimizes service disruption for planned upgrades in cellular networks by proactively tuning network

configuration. Mercury [20] time-aligns multiple network upgrades and conducts an aggregate performance impact analysis. PRISM [18] is a real-time analytics tool to detect performance degradations immediately after a planned maintenance. Litmus [19] uses robust algorithms to assess the performance impact of network changes by comparing study group with control group. However, Mercury, PRISM and Litmus cannot perform root-cause analysis.

**Network and service diagnosis:** [15] develops an open platform to monitor LTE radio performance by extracting information from the LTE control channel. A number of works, such as ThinkRF [31] and IntelliJudge [27], sniff RF signals for testing inter-operability. [10] studies the interactions between applications, transport protocol, and the radio layer in a large LTE network in US. It reports 52.6% of TCP flows are throttled by TCP receive window. [32] uncovers several problematic interactions among the control-plane protocols in cellular networks. [28], [36], [2] use machine learning techniques to diagnose problems in networks. Argus [35] uses Holt-Winters algorithm to detect and localize service performance anomalies in large ISP networks. NICE [21], WISE [30], Giza [17], NetMedic [12], GRCA [34], and URCA [29] capture the statistical dependencies between network and service performance. Spectroscope [26] and X-ray [1] compare two executions of program before and after the change to diagnose performance changes. DiffProv [4] uses differential provenance to compare good and bad examples of system behavior to identify the reason for the bad behavior. As also pointed by the authors, DiffProv assumes network behavior is deterministic which is often not the case in operational cellular network.

**Remarks:** Our work complements existing work by focusing on a new aspect of network management – selecting test locations and contrasting performance analysis, which is a critical but under-explored problem.

## VI. Conclusion

Cellular networks frequently go through upgrades. It is important to automatically design field tests to identify the features that affect the upgrade performance and predict the performance of untested location and diagnose the performance issues during the upgrades. We develop algorithms to automate both processes, and apply them to the real traces from a major cellular provider. We show that by sampling 1-1.5% of eNodeBs in real world case studies, we can identify the root cause with close to 100% accuracy. Encouraged by the promising performance, we are working with the network operation team to incorporate our solution to their operation.

**Acknowledgment:** We thank Chris Hristov, Jia Wang, Jennifer Yates, Chris Rice, Jiasi Chen, and anonymous reviewers for their insightful feedback on the paper. We strongly appreciate the collaboration and continuous support from the AT&T Network Engineering and Operations teams in the application of *Reflection*, regular feedback to improve its usability, and case-study analysis. This work is supported in part by NSF Grant CNS-1718089.

## References

[1] M. Attariyan, M. Chow, and J. Flinn. X-ray: Automating root-cause diagnosis of performance anomalies in production software. In *USENIX OSDI*, 2012.
[2] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. Developing a predictive model of quality of experience for internet video. In *Proc. of ACM SIGCOMM*, 2013.
[3] A. Barg, A. Mazumdar, and R. Wang. Restricted isometry property of random subdictionaries. *IEEE Trans. on Information Theory*, 2015.
[4] A. Chen, Y. Wu, A. Haeberlen, W. Zhou, and B. T. Loo. The good, the bad, and the differences: Better network diagnostics with differential provenance. In *ACM SIGCOMM*, 2016.
[5] Y. Chen, D. Bindel, H. Song, and R. H. Katz. An algebraic approach to practical and scalable overlay network monitoring. In *ACM SIGCOMM*, 2004.
[6] S. Fischer. Observed Time Difference Of Arrival (OTDOA) Positioning in 3GPP LTE. https://www.qualcomm.com/media/documents/files/otdoa-positioning-in-3gpp-lte.pdf.
[7] S. Ghorbani and M. Caesar. Walk the line: Consistent network updates with bandwidth guarantees. In *Proc. of HotSDN*, 2012.
[8] M. Grindal, J. Offutt, and S. F. Andler. Combination testing strategies: A survey. *Software Testing, Verification, and Reliability*, 2005.
[9] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven wan. In *Proc. of ACM SIGCOMM*, 2013.
[10] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, and O. S. Subhabrata Sen. An in-depth study of LTE: effect of network protocol and application behavior on performance. In *ACM SIGCOMM*, 2013.
[11] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer. Dynamic scheduling of network updates. In *Proc. of ACM SIGCOMM*, 2014.
[12] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl. Detailed diagnosis in enterprise networks. In *ACM SIGCOMM*, 2009.
[13] R. Kohavi, A. Deng, B. Frasca, T. Walker, Y. Xu, and N. Pohlmann. Online controlled experiments at large scale. In *ACM KDD*, 2013.
[14] R. Kohavi, R. M. Henne, and D. Sommerfield. Practical guide to controlled experiments on the web: listen to your customers not to the hippo. In *Proc. of ACM KDD*, 2007.
[15] S. Kumar, E. Hamed, D. Katabi, and L. E. Li. LTE radio analytics made easy and accessible. In *Proc. of SIGCOMM*, 2014.
[16] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz. zupdate: Updating data center networks with zero loss. In *ACM SIGCOMM*, 2013.
[17] A. Mahimkar, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and Q. Zhao. Towards automated performance diagnosis in a large IPTV network. In *Proc. of ACM SIGCOMM*, 2009.
[18] 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 *Proc. of ACM CoNEXT*, 2011.
[19] A. Mahimkar, Z. Ge, J. Yates, C. Hristov, V. Cordaro, S. Smith, J. Xu, and M. Stockert. Robust assessment of changes in cellular networks. In *ACM CoNEXT*, 2013.
[20] A. Mahimkar, H. H. Song, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and J. Emmons. Detecting the performance impact of upgrades in large operational networks. In *Proc. of ACM SIGCOMM*, 2010.
[21] A. Mahimkar, J. Yates, Y. Zhang, A. Shaikh, J. Wang, Z. Ge, and C. T. Ee. Troubleshooting chronic conditions in large IP networks. In *Proc. of ACM CoNEXT*, 2008.
[22] M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques*. Springer, 1978.
[23] C. Nie and H. Leung. A survey of combinatorial testing. *ACM Comput. Surv.*, 43(2):11:1–11:29, Feb. 2011.
[24] PSU. Psu. https://onlinecourses.science.psu.edu/stat414/node/268.
[25] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *Proc. of ACM SIGCOMM*, 2012.
[26] R. R. Sambasivan and A. X. Z. et al. Diagnosing performance changes by comparing request flows. In *USENIX NSDI*, 2011.
[27] The wavejudge wireless test system. http://www.sanjole.com/our-products/wavejudge-test-system/.
[28] M. Z. Shafiq, L. Ji, A. Liu, J. Pang, S. Venkataraman, and J. Wang. A first look at cellular network performance during crowded event. In *Proc. of ACM SIGMETRICS*, 2013.
[29] F. Silveira and C. Diot. URCA: Pulling out anomalies by their root causes. In *INFOCOM*, 2010.
[30] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar. Answering what-if deployment and configuration questions with WISE. In *ACM SIGCOMM*, 2008.
[31] Product concept brief for distributed and remote LTE analysis. http://thinkrf.com/.
[32] G.-H. Tu, Y. Li, C. Peng, C.-Y. Li, H. Wang, and S. Lu. Control-plane protocol interactions in cellular networks. In *Proc. of SIGCOMM*, 2014.
[33] X. Xu, I. Broustis, Z. Ge, R. Govindan, A. Mahimkar, N. Shankaranarayanan, and J. Wang. Magus: Minimizing cellular service disruption during network upgrades. In *ACM CoNEXT*, 2015.
[34] H. Yan, L. Breslau, Z. Ge, D. Massey, D. Pei, and J. Yates. G-RCA: a generic root cause analysis platform for service quality management in large ip networks. In *ACM CoNEXT*, 2010.
[35] H. Yan, A. Flavel, and Z. G. et al. Argus: End-to-end service anomaly detection and localization from an ISP's point of view. In *IEEE INFOCOM*, 2012.
[36] S. Zhou, J. Yang, D. Xu, G. Li, Y. Jin, Z. Ge, M. B. Kosseifi, R. Doverspike, Y. Chen, and L. Ying. Proactive call drop avoidance in umts networks. In *Proc. of IEEE INFOCOM*, 2013.