Detecting and Diagnosing Performance Impact of Smartphone Software Upgrades

Ajay Mahimkar
AT&T

***Abstract*—Smartphone manufacturers often release software upgrades to their users for improving service performance, patching security vulnerabilities, enhancing device stability, fixing bugs, increasing battery life, or even enriching the graphical user interface. It is crucial to monitor the smartphones after software upgrades to either confirm their expected impacts, or quickly identify any undesirable behaviors. In this paper, we focus on automatically detecting the software upgrades on smartphones and analyzing their service performance impacts. The complex interactions between the software on the smartphones and the cellular networks make it hard to differentiate if the impacts are smartphone-centric, or network-centric. We propose a new approach, SSM (Smartphone Specific Monitoring) for conducting pre/post impact analysis of multiple service performance metrics across smartphones aggregated by their type, make, model and network locations. Using one-year worth of operational network data, we demonstrate the effectiveness of SSM in accurately detecting and diagnosing the performance impact of smartphone software upgrades.**

I. Introduction

In recent years, we have witnessed a phenomenal rise in smartphone popularity and usage with people from all parts of the society glued to their smartphones for communication, entertainment, social network activity, and business needs. There is a wide variety of smartphones and applications running on top of them that introduce several challenges for the service providers as they continue to offer excellent quality of experience to hundreds of millions of smartphone users. These challenges will continue to surmount with introduction of new types of smartphones, Internet of Things (for example, connected cars, digital homes, connected cities, connected health), and machine-to-machine (M2M) communications.

The software that runs on these smartphones and supports a diverse set of applications, plays an important role in providing a nice graphical user interface, managing the battery life, and offering excellent service quality of experience to its users. It also interacts with the underlying cellular networks in creating a seamless mode of experience and handling user mobility, changing traffic demands, and application requirements. The cellular networks are changing at an extremely rapid pace - technology evolution from GSM to UMTS to LTE, support for multiple services such as voice, video and data, and network virtualization. The dynamic nature of both the smartphones and the cellular networks and their complex interactions make the service quality management extremely challenging.

The smartphone manufacturers release new hardware models and software versions to their subscribers for an improved quality of experience. We will use the terms smartphone *user* and smartphone *subscriber* interchangeably. The cellular service providers and the smartphone manufacturers have to carefully monitor these releases and their service performance impacts. If there is an unexpected degradation, then either the smartphone manufacturer has to fix it through a subsequent software release, or the cellular service provider has to update its network configuration. Pinpointing the root-cause is important for the accurate resolution of the issue.

In this paper[1], we focus on the software upgrades[2] implemented by the smartphone subscribers and their service performance impacts. The smartphone manufacturers expect the service performance to improve (or, at-least not degrade) after the software upgrade because the intention is to either patch security vulnerabilities, resolve software bugs on the smartphones, enhance their stability, improve battery life, or even enrich the graphical user interface. Besides extensive laboratory testing, it is important to capture the service performance impacts in the field because a controlled laboratory environment will not be able to emulate realistic network conditions and interactions with the new software on the smartphone. Hence, a small-scale testing is conducted in the field before the software is made available to all the smartphone subscribers. The large scale and heterogeneity of the cellular network makes it extremely hard to enumerate all possible configuration interactions. Thus, the cellular service providers and smartphone manufacturers monitor the performance impacts as the software is rolled out across all smartphone subscribers.

Our goal is to start with smartphone software upgrades and conduct a pre/post impact analysis of service performance to capture their impacts. There are several technical challenges to be addressed in order to design an effective solution. We make the following observations based on data collected from the cellular network: (i) **Scale:** There are on the order of hundreds of millions of smartphone subscribers and tracking software upgrades and their service performance impact at such a large scale is non trivial. (ii) **Heterogeneity:** There is a wide variety of smartphone classes and their software versions. We use the type, make (or, manufacturer) and model to group smartphone subscribers into classes. An example type, make and model is a smartphone that is LTE-capable (type), Apple (make) iPhone 6S (model). In the rest of the paper, we will use the terms make and manufacturer interchangeably. The unique number of smartphone type, make, model and software version is on the order of thousands. (iii) **Staggered software**

[1]No personally identifiable information (PII) was gathered or used in conducting this study. To the extent any data was analyzed, it was anonymous and/or aggregated data.

[2]Software upgrade includes both application upgrade and firmware upgrade. In the context of this paper, the software upgrade denotes the firmware upgrade.

978-3-901882-85-2 © 2016 IFIP

roll-out: Once the smartphone manufacturer releases the new software, not all subscribers would download and install the software at the same time. This creates a staggered view of the software roll-out or deployment across subscribers and makes the impact analysis hard. (iv) **Subscriber penetration:** Some smartphone type, make and model are inherently more popular than others. We define subscriber penetration as the number of subscribers within the type, make and model compared to the global subscriber population. We observe a different degree of subscriber penetration across different type, make and model that makes performance comparisons hard.

Our approach and contributions. We propose a new approach SSM (Smartphone Specific Monitoring) to automatically detect software upgrades on the smartphones and their performance impacts. We use call detail records to capture attributes of smartphone subscribers (type, make, model and software version), and call statistics (such as normal termination status or failure, time spent on network technologies). To handle the extremely large number of subscribers, we aggregate the call detail records grouped by smartphone hierarchy of type, make and model and determine the upgrade time as the time when we have sufficient subscriber penetration on the new software version. Given the upgrade time, SSM would compare the service performance before and after at the aggregation of type, make and model to detect if there is an improvement, degradation or no impact due to the upgrade. Note here, we aggregate to the smartphone type, make and model and do not conduct pre/post performance comparison across individual subscribers or software version. This is because the performance metrics at the subscriber-level could be highly variable for any meaningful analysis and comparing pre/post at the software version-level has the issue of different subscriber penetration leading to different performance profiles. By operating at the right aggregation level in the smartphone hierarchy, SSM addresses the staggered roll-out of the smartphone software upgrades.

We propose to use a recursive cumulative sum approach in SSM to statistically compare the performance before and after the smartphone software upgrade to detect the impact, and use measures such as mean, median, or standard deviation to label the performance impact as an improvement or a degradation. To handle the heterogeneity of smartphone and network configuration, and their complex interactions, we design an impact diagnosis approach in SSM to categorize the impacts as *network-centric*, *smartphone-and-network interactions*, and *smartphone-centric*. An impact is labeled as *network-centric* if both the upgraded and the non-upgraded smartphones have similar performance impacts around the software upgrade time. An impact is labeled as having *smartphone-and-network interactions* if only specific locations in the network have the performance impact on the upgraded smartphones. An impact is *smartphone-centric* if it is network-wide and observed only for the upgraded smartphones. Our goal in this paper is to identify smartphone-centric impacts of software upgrades. The diagnosis approach filters out the network-centric and smartphone-and-network interactions and accurately identifies the impact of smartphone software upgrades.

We present a thorough evaluation of SSM using one year worth of operational data collected from a large cellular service provider (Section V). Our results demonstrate the effectiveness of SSM in the accurate classification of network-centric impacts, smartphone-and-network interactions, and smartphone-centric impacts. Encouraged by the successful application of SSM, we are working on deploying it in production environments.

Remarks. The SSM solution perspective is from a cellular service provider. The smartphone manufacturers do not have visibility into performance experienced by the users with smartphones belonging to other manufactures and hence cannot accurately capture if the impact is primarily due to the smartphone software, or due to events in the cellular network. Impacts due to events in the network are false associations from the smartphone manufacturer perspective because the resolution of the issue should come from the network and apply to all smartphones. The cellular service provider is in a better position to differentiate the impacts due to network versus smartphone software and can notify the smartphone manufacturer of the *smartphone-centric* impacts. Depending on the root-cause analysis, the smartphone manufacturer can decide to resolve in a subsequent software release. Our approach benefits (a) the cellular service provider by not requiring them to invest resources in resolution of the impacts caused primarily by the smartphone software, and (b) the smartphone manufacturers by not requiring them to invest resources in resolution of the impacts caused by the network.

II. Capturing Smartphone Software Upgrades

Our goal is to capture the software upgrades on the smartphone subscribers along with the time of the upgrade. The cellular service provider maintains a database of subscriber information such as smartphone type, make, model, and operating system version, date of activation, and voice/data service plan details. We use ***call detail records (CDRs)*** collected by the cellular service provider to detect software upgrades on the smartphones. CDRs contain a record of each voice call and data session that the smartphone subscriber places on the cellular network. They are collected in real-time at the core switches in the network (Mobile Switching Center for voice CDRs and Session Gateway for data CDRs).

The voice and data CDRs provide a variety of information including connection times, duration, originating and terminating IMSI (International Mobile Subscriber Identity), termination status (a call can be terminated normally by the subscriber, or abnormally terminated by the network resulting in a blocked or a dropped call), originating number IMEI (International Mobile Station Equipment Identity), and a list of cells (base stations) utilized by the call. The IMEI information is used to map the subscriber to smartphone type, make, model and operating system version. The IMEI consists of 16 digits - the first 8 digits represent the Type Allocation Code (TAC). TAC is used to identify the type, make, and model of the smartphone. The last 2 digits represent the operating system version installed on the smartphone. A change in the last 2 digits of the IMEI between CDRs for a subscriber indicates a software version upgrade. This approach of tracking software version upgrades has tremendous scalability challenges because the number of CDRs in a 5-minute time-window can easily exceed

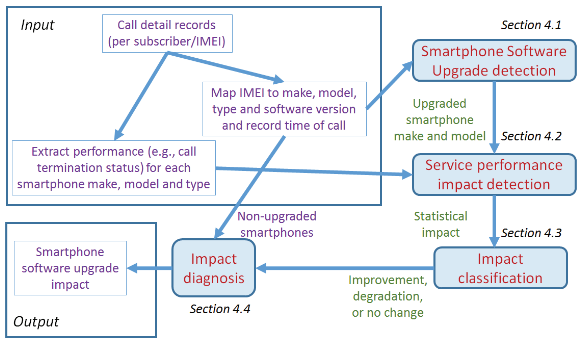
Input
Call detail records (per subscriber/IMEI)
Map IMEI to make, model, type and software version and record time of call
Extract performance (e.g., call termination status) for each smartphone make, model and type
Section 4.1
Smartphone Software Upgrade detection
Upgraded smartphone make and model
Section 4.2
Service performance impact detection
Statistical impact
Section 4.3
Impact classification
Improvement, degradation, or no change
Non-upgraded smartphones
Impact diagnosis
Section 4.4
Smartphone software upgrade impact
Output

Fig. 1. Overview of SSM workflow. SSM uses CDRs as the input to detect smartphone software upgrades and service performance metrics derived from CDR to output their impacts.

hundreds of millions of records. Section IV-A will describe our scalable approach to detect smartphone software upgrades.

III. Capturing Service Performance

For a given software upgrade on a smartphone, we would like to identify the list of service performance metrics that need to monitored. We again use call detail records that provide the status of termination (either subscriber-triggered or network-induced), and list of cells it traverses (the voice call or data session can either be on LTE, UMTS or GSM network). We define a blocked call as a failed connection attempt, and a dropped call as an abnormal termination by the network. Both blocked and dropped calls can either be due to issues in the radio access network (referred to as RIF issues), or due to issues in the core network. We use the list of cells to identify the terminating cell and the technology type (LTE, UMTS or GSM). The call then gets labeled as successful termination, blocked/dropped RIF (at radio access network) or non-RIF (at core) and at specific technology. For example, VoLTE RIF captures the dropped voice calls in the LTE network.

Ideally, we require all LTE capable smartphones to use the LTE network. But sometimes, they might fall-back to either UMTS or GSM network because of lack of coverage, overload condition, software bug, or hard failure. Such scenarios however, should be rare. Hence, we measure the number of calls not on LTE (CNOV) and time not on LTE (TNOL) for LTE capable smartphones.

IV. Design and Implementation

In this section, we present the design and implementation of SSM. Figure 1 gives an overview of the SSM workflow. SSM continuously monitors the call detail records and identifies the software upgrades on smartphones (Section IV-A). These upgrades serve as trigger for the pre/post performance impact detection (Section IV-B). By comparing the service performance before and after the time of the software upgrade, SSM classifies the impact as either improvement, degradation, or no change (Section IV-C). Finally, SSM compares these impacts across network locations and non-upgraded smartphone to accurately diagnose the impact of upgrades (Section IV-D).

A. Upgrade Detection

As described in Section II, we use the subscriber IMEI information in each call detail record to infer the smartphone make, model, type and software version. Detecting software upgrades on smartphones requires us to track changes in the software version for each subscriber. This operation can be computationally expensive and require a huge amount of storage to handle hundreds of millions of subscribers. We propose a simple yet effective approach for scalable detection of software upgrades on smartphones. Recall that our high level goal is to identify the performance impact of a software upgrade across smartphone make and model, and not for individual subscribers. We can thus aggregate the CDR information and conduct the impact analysis for a group of subscribers belonging to a smartphone make and model.

We use the mapping TAC $\rightarrow \{$make, model, type$\}$ to map the subscribers to the make, model and type of smartphones. By combining the TAC with software version, we compute the number of subscribers on a particular software version for a given smartphone make, model and type. We store the number of subscribers on a periodic basis within the aggregation hierarchy. Figure 2 shows the aggregation hierarchy for mapping from IMEI to software version, model, make and type. We thus have a time-series of the number of subscribers on a software version. Dividing it by the total number of subscribers within the smartphone make, model and type, we compute the subscriber penetration for a smartphone make, model, type, and software version. The computation of the number of subscribers for each smartphone make, model, type, and software version aggregation is light-weight as compared to tracking the changes for each subscriber. The storage is also considerably lower because we now have to store the time-series only at the aggregated levels.

For detecting upgrades, we mine the subscriber penetration time-series and look for increasing trends starting from zero or a very low penetration value and ramping up. The maximum value for subscriber penetration is 100% indicating that all subscribers for the aggregation are on the software version. We constrain the ramp-up to begin from zero. Else, we might inaccurately discover the intermediate time of the software roll-out as the start of the upgrade. Typically, the subscribers would catch up to the new software versions within a few days (given the push notifications from smartphone manufacturers). For impact analysis, we would need to identify the date of upgrade around which one can compare performance before and after. If we pick the upgrade date too soon, then we have a smaller subscriber penetration making it hard to conduct meaningful statistical analysis. A date too far has the issue of contaminated baseline for the before interval. Thus, it is crucial to accurately identify the date of software upgrade.

We use a threshold-based approach to identify the date for the software upgrade corresponding to a smartphone make, model, type and software version. Once the subscriber penetration reaches the threshold, the date would serve as the trigger for pre/post impact analysis. We select a moderately low value of 10% as the threshold d so that the pre upgrade interval can capture the performance behaviors with majority of subscribers on the older software version and the post upgrade interval can capture the performance behaviors with majority of subscribers on the newer software version. For smartphone make, model, type and software version with high subscriber penetration, we have a sufficient number of samples (or, subscribers) for

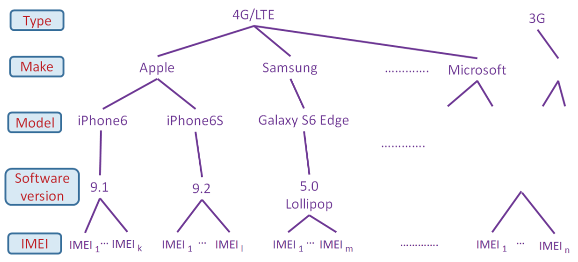

Fig. 2. Aggregation hierarchy used in SSM. The IMEIs can be on the order of hundreds of millions.

statistical analysis. For very low subscriber penetration, the ramp up should be fast (within a few days) because of low number of total subscribers. However, if the ramp up is slow, we will adopt the impact detection approach to adjust for accurate impact analysis.

Note that SSM primarily uses the software version information from IMEI to track smartphone software upgrades. In some cases, the users can modify their smartphone software by updating with a customized ROM. The customized ROM does not follow the numbering convention in terms of IMEI, which makes it difficult to determine the right software version using our approach. In the future, we will explore alternative methods to determine the smartphone software version without relying on IMEI.

B. Impact Detection

Given the trigger time of the smartphone software upgrade, our goal is to detect if there is a statistical change in behavior in its service performance. If the service performance behavior changes around the smartphone software upgrade in an unexpected fashion, then it is important to the cellular service provider and the smartphone manufacturer for further investigation. We again use the call detail records for each smartphone to compute the service performance metrics aggregated by the smartphone make, model and type. We use the same aggregation hierarchy as shown in Figure 2. This approach is scalable across a large number of subscribers (hundreds of millions) and an extremely large number of call detail records (hundreds of millions of termination status of calls within 5 minutes). Also, note that our approach is a one-pass parsing of the call detail records to construct subscriber penetration and service performance aggregates across smartphone make, model and type. There is a subtle difference between the aggregates - subscriber penetration is stored at $\{$make, model, type, software version$\}$ and service performance is stored at $\{$make, model, type$\}$. This is to enable creation of the baseline performance time-series data.

In order to deal with variability in the service performance time-series, we use a recursive cumulative sum (CUSUM) algorithm to identify multiple change-points. Recursive cumulative sum is good at detecting level shifts and ramp up or down behaviors [15]. We use a sufficiently long duration for creating the pre and post interval performance time-series. This eliminates any transient unrelated change-points (for example, point anomalies and spikes). For each change-point c_t, we note features such as time of behavior change, time of previous change-point c_{t-1} and subsequent one c_{t+1}, statistical measures such as mean, median, and deviation for before interval ($t-1$ to t), and after interval (t to $t+1$). We now iterate through all change-points to identify the nearest co-occurring smartphone upgrade time. If the time difference between the performance change-point and smartphone upgrade is less than a threshold T, then we label it as having a statistical performance impact of software upgrade.

We adjust the threshold T based on the subscriber penetration of the smartphone make, model, type and software version. We propose an adaptive mechanism to control the correlation time-window between the smartphone software upgrade captured across the make, model and type, and the performance impact observed across all smartphones for that make, model and type. For high subscriber penetration, we expect the performance impact to be visible immediately after the upgrade time detected using our approach described in Section IV-A and hence we set a lower threshold value for T. We use $T = +/-3$ days in our evaluation. For very low subscriber penetration, if the ramp up on upgrade is fast, then we can use the same threshold above. But, we adapt it to a higher value for very slow ramp ups for low subscriber penetration (we use $T = +/-9$ days).

C. Impact Classification

Once our impact detector identifies a statistical change in performance behavior around the smartphone software upgrade, the next step is to identify if the behavior change corresponds to an improvement, degradation, or no impact. This classification is important and useful to capture the expected or unexpected behaviors of smartphone software upgrades. For example, if the smartphone manufacturer resolves any issues in previous software versions, then our expectation is a performance improvement in the new version compared to the old. For the co-occurring change-point around the smartphone software upgrade, we use the statistical measures before and after the upgrade interval to label the service performance as an increase (high value in the after interval as compared to the before), decrease or no impact. Depending on the type of service performance metric, we then label it as improvement, degradation, or no impact. For example, increase in VoLTE RIF percentage indicates a performance degradation because it corresponds to an increase in the number of VoLTE dropped calls by the network. On the other hand, increase in data throughput indicates a performance improvement.

At this point, we know the performance impact labeled with the software upgrade on the smartphones. One has to use caution because the co-occurrence can merely be coincidental. False co-occurrences can occur due to (a) the performance impact for smartphone software upgrade can overlap with a network-wide impact induced by a network upgrade, (b) the performance impact can be different across different network locations and cancel out when aggregating at the smartphone make, model and type - this can happen due to conflicting interactions between network software and smartphone software, or (c) external factors such as holiday season, foliage changes, traffic pattern changes can impact service performance across multiple smartphones and not just the ones that had the

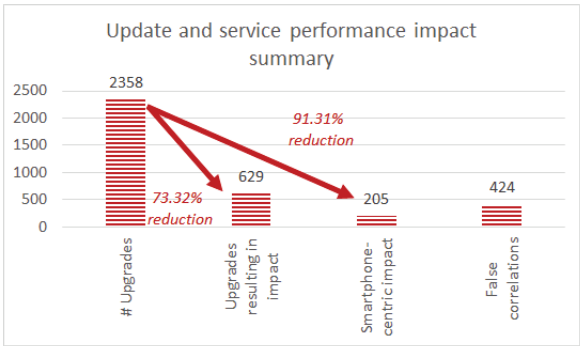

Fig. 3. Upgrade and service performance impact summary.

software upgrade. We tackle false co-occurrences using our impact diagnosis approach described in the next sub-section.

D. Impact Diagnosis

We devise an impact diagnosis approach to eliminate impacts due to network-centric behaviors and interactions between software on the smartphone and the network. We call the true co-occurrences as *smartphone-centric* performance impacts of smartphone software upgrades. We append the aggregation hierarchy in Figure 2 by incorporating network location information. We run our impact detection and classification approaches not only for the upgraded smartphones but also for the non-upgraded ones at these network locations. We use the following rules to eliminate false co-occurrences:

1. If similar performance impacts (for example, improvement) are observed across both upgraded as well as non-upgraded smartphones, then we label it as *network-centric*. This tackles the network-wide impacts and external factors. We then eliminate network locations in the network aggregation that have similar performance impacts across upgraded and non-upgraded smartphones.
2. For the remaining locations, we label them with the performance impact for the upgraded smartphones in that location. We count the number of network locations with performance improvement versus degradation and compare their difference to a threshold K. If the two numbers are close to each other (less than K), then we label the performance impact as induced by *smartphone-and-network interactions*. K can be determined using a learning approach based on historical data that captures the impacts observed across multiple network locations for the upgraded smartphones. We observed that the difference between the number of network locations with improvement and those with degradation is less than 2 for scenarios where the smartphone-and-network interactions at specific locations led to a contrasting impact across locations. We thus set $K = 2$ in our setup.

The final impact of the smartphone software upgrade is determined by the majority of the network locations with similar labels of performance impact. These rules though simple, are powerful enough to accurately identify the true performance impact of smartphone software upgrades.

Upgrade detection	Impact detection	Impact classification	Impact diagnosis
43 seconds	4 hours	164 seconds	51 seconds

TABLE I
EXECUTION SPEED OF SSM TO ANALYZE ONE YEAR WORTH OF DATA.

V. EVALUATION

We use one-year worth of data collected from operational cellular network to thoroughly evaluate SSM.

A. Results

Figure 3 provides a summary of the smartphone software upgrades and their service performance impacts captured using SSM. Out of 2358 software upgrades across different smartphone make, model and type, we observe that 629 result in a statistical impact across at-least one of the service performance metrics. From a monitoring perspective, the impact detection in SSM achieves a 73.32% reduction in the number of software upgrades that the operations teams would have to look at. After conducting impact classification and diagnosis, SSM eliminates 424 false co-occurrences. This results in 91.31% reduction - a significant one from the operations team perspective and efficient utilization of their scarce time resources in zooming into the 205 smartphone-centric ones.

Table I shows the execution speed of SSM in analyzing one year worth of data. The daily summary of the CDR data is computed using a separate process and with a daily cron job. Given the daily CDR data, SSM took around 43 seconds to detect the smartphone software upgrades. The impact detection took the maximum (around 4 hours) because of the application of the recursive CUSUM across smartphone type, make, model and the network locations. The correlation of the impact with upgrade time and the impact classification took around 164 seconds. Finally, the impact diagnosis took around 51 seconds. In summary, SSM took around 4 hours to capture the smartphone software upgrade impact after analyzing one year worth of data.

B. Operational Case Studies

In this section, we present two operational case studies that we validate using SSM. All of the case studies were historically analyzed by the operations teams of the cellular service provider and incurred a significant time overhead (on the order of days) in collating the smartphone aggregated data and analyzing the impacts of the software upgrades. SSM however is able to scalably and quickly mine the large-scale data and accurately identify the performance impacts.

Case study I: In our first case study, we used SSM to identify a minor performance degradation in VoLTE RIF as the software version was upgraded by the smartphones from version X to version Y. At the time of the degradation, the service operations teams had notified the smartphone software manufacturer and it was confirmed as a software bug in version Y. The smartphone manufacturer fixed the issue in software version Z. We used SSM to confirm that the VoLTE RIF improved as the software upgraded from Y to Z. As can be seen in figure 4, VoLTE RIF first increases after first software upgrade (X $\rightarrow$ Y). The increase in VoLTE RIF indicates a higher number of voice over LTE call failures in the post upgrade interval as compared to the pre interval. With software

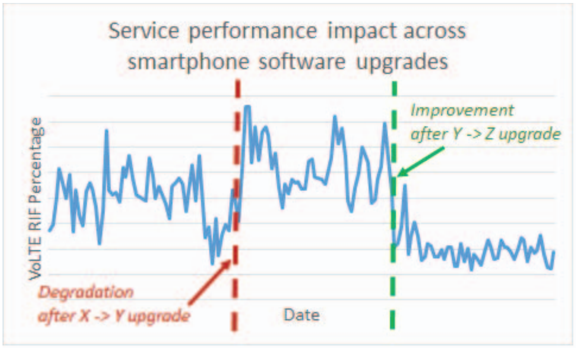

Fig. 4. Case Study I: Performance impact of smartphone software upgrades on Voice over LTE (VoLTE) call failures due to RIF (radio access network). The first software upgrade (X $\rightarrow$ Y) led to minor performance degradation which was resolved in the next software release (Y $\rightarrow$ Z).

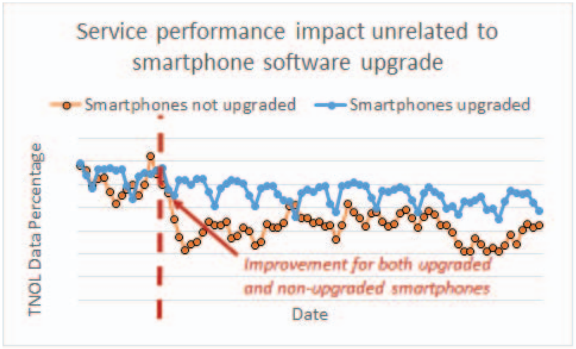

Fig. 5. Case Study II: TNOL (Time Not on LTE) data performance improvement observed across both upgraded as well as non-upgraded smartphones. This indicates a false co-occurrence that is accurately captured by SSM.

version upgrade from Y to Z, we notice an expected improvement (decrease) in VoLTE RIF that matches the expectation of the cellular service provider and the smartphone manufacturer.

Case study II: Our second case study is an interesting validation example using SSM. Figure 5 shows performance improvement in TNOL (Time not on LTE) data not only for the smartphones that had the software upgrade, but also for the non-upgraded smartphones. A decrease in TNOL data after the software upgrade indicates that more time is spent on LTE which is desirable, and hence considered as a performance improvement. If someone had looked at these impact agnostic of the performance behaviors of the non-upgraded smartphones, then they would have inaccurately concluded the improvements. By looking across the spectrum, SSM accurately captures no performance impact of the upgrade.

VI. Related Work

Software upgrade impact analysis. Mercury [15] uses time-alignment and aggregate change detection to provide a holistic view of the performance impact after network upgrades. PRISM [13] is a real-time detection tool and uses robust singular value decomposition to detect performance anomalies immediately after planned maintenance activities. FUNNEL [25] conducts rapid and robust impact assessment of software changes deployed in large Internet-based services. It uses Difference-in-Difference (DiD) approach to conduct a relative comparison of performance between upgraded and non-upgraded servers. Spectroscope [19] and X-ray [2] compare two executions of program before and after the change to diagnose performance changes. Litmus [14] uses robust regression tests to compare performance between study group (network elements with upgrade) and control group (network elements without the upgrade) and quantify the performance impact on the study group. A/B testing [8], [9] are used in the web domains for assessing the impact of new feature releases. None of the above approaches were aimed at the problem scope of identifying the impact of smartphone software upgrades. SSM tackles the large scale of the number of smartphone subscribers (on the order of hundreds of millions) by analyzing the impacts at the right aggregation hierarchy.

Service diagnosis. Argus [24] uses Holt-Winters based forecasting to detect and localize end-to-end service anomalies in large ISP networks. ASTUTE [21] is a traffic anomaly detector that leverages equilibrium across flows and correlation across anomalies. PCA [6], [10], [11], [18] is used to detect network-wide anomalies for origin-destination traffic matrices. [26] uses spatio-temporal compressive sensing to detect anomalies in traffic matrices. Sherlock [3] proposes a multi-level graph inference to discover the service-level dependencies in enterprise networks and Orion [5] uses delay timing analysis to discover traffic dependencies. [1] uses IP flow data to diagnose end-to-end performance of the mobile Internet. NICE [16], WISE [22], Giza [12], NetMedic [7], GRCA [23], and URCA [20] use statistical data mining to identify dependencies between network and service performance metrics. QProbe [4] uses active and controlled network measurement to localize the performance bottleneck in cellular networks. ABSENCE [17] uses aggregated customer usage data to detect service disruptions in cellular networks. Its focus is on customer usage whereas, SSM focuses on multiple service performance metrics on the smartphones (such as dropped calls due to issues in the radio access networks, time not spent on LTE for LTE capable smartphones), and impacts only induced by the smartphone software upgrades.

VII. Conclusions

In this paper, we presented the design and implementation of SSM, for detection and diagnosis of impacts of smartphone software upgrades. SSM uses call detail records to automatically detect changes in software versions on the smartphones. It then conducts a pre/post impact analysis on the service performance metrics and identifies if there is an improvement, degradation, or no impact around the software upgrade. By correlating impacts across network locations and non-upgraded smartphones, SSM accurately diagnoses the performance impacts. We used one-year worth of operational network data to demonstrate the effectiveness of SSM.

Acknowledgement

We thank Aman Shaikh, Zihui Ge, Jennifer Yates, Rizwan Muzaffar, our shepherd Michele Nogueira and the CNSM anonymous reviewers for their insightful feedback on the paper. We acknowledge the support from the Mobility Network Operations and Engineering teams in validation of SSM and case study analysis.

REFERENCES

[1] C. Amrutkar, M. Hiltunen, T. Jim, K. Joshi, O. Spatscheck, P. Traynor, and S. Venkataraman. Why is my smartphone slow? on the fly diagnosis of underperformance on the mobile internet. In *DSN*, 2013.

[2] M. Attariyan, M. Chow, and J. Flinn. X-ray: Automating root-cause diagnosis of performance anomalies in production software. In *USENIX OSDI*, 2012.

[3] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *ACM SIGCOMM*, 2007.

[4] N. Baranasuriya, V. Navda, V. N. Padmanabhan, and S. Gilbert. QProbe: Locating the bottleneck in cellular communication. In *ACM CONEXT*, 2015.

[5] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl. Automating network application dependency discovery: Experiences, limitations, and new solutions. In *USENIX OSDI*, 2008.

[6] Y. Huang, N. Feamster, A. Lakhina, and J. J. Xu. Diagnosing network disruptions with network-wide analysis. In *ACM SIGMETRICS*, 2007.

[7] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl. Detailed diagnosis in enterprise networks. In *ACM SIGCOMM*, 2009.

[8] R. Kohavi, A. Deng, B. Frasca, R. Longbotham, T. Walker, and Y. Xu. Trustworthy online controlled experiments: five puzzling outcomes explained. In *ACM KDD*, 2012.

[9] 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 *ACM KDD*, 2007.

[10] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM*, 2004.

[11] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM*, 2005.

[12] 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 *ACM SIGCOMM*, 2009.

[13] 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 *ACM CoNEXT*, 2011.

[14] 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.

[15] 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 *ACM SIGCOMM*, 2010.

[16] A. Mahimkar, J. Yates, Y. Zhang, A. Shaikh, J. Wang, Z. Ge, and C. T. Ee. Troubleshooting chronic conditions in large IP networks. In *ACM CoNEXT*, 2008.

[17] B. Nguyen, Z. Ge, J. V. der Merwe, H. Yan, and J. Yates. ABSENCE: Usage-based failure detection in mobile networks. In *ACM MOBICOM*, 2015.

[18] H. Ringberg, A. Soule, J. Rexford, and C. Diot. Sensitivity of PCA for traffic anomaly detection. In *ACM SIGMETRICS*, 2007.

[19] R. R. Sambasivan, A. X. Zheng, M. D. Rosa, E. Krevat, S. Whitman, M. Stroucken, W. Wang, L. Xu, and G. R. Ganger. Diagnosing performance changes by comparing request flows. In *USENIX NSDI*, 2011.

[20] F. Silveira and C. Diot. URCA: Pulling out anomalies by their root causes. In *IEEE INFOCOM*, 2010.

[21] F. Silveira, C. Diot, N. Taft, and R. Govindan. ASTUTE: Detecting a different class of traffic anomalies. In *SIGCOMM*, 2010.

[22] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar. Answering what-if deployment and configuration questions with WISE. In *ACM SIGCOMM*, 2008.

[23] 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.

[24] H. Yan, A. Flavel, Z. Ge, A. Gerber, D. Massey, C. Papadopoulos, H. Shah, and J. Yates. Argus: End-to-end service anomaly detection and localization from an ISP's point of view. In *IEEE INFOCOM*, 2012.

[25] S. Zhang, Y. Liu, D. Pei, Y. Chen, X. Qu, S. Tao, and Z. Zang. Rapid and robust impact assessment of software changes in large internet-based services. In *ACM CoNEXT*, 2015.

[26] Y. Zhang, M. Roughan, W. Willinger, and L. Qiu. Spatio-temporal compressive sensing and internet traffic matrices. In *ACM SIGCOMM*, 2009.