An Empirical Study of Mobile Network Behavior and Application Performance in the Wild

Shiwei Zhang
Southern University of Science and Technology, P. R. China
Peng Cheng Laboratory, P. R. China
zhangsw@mail.sustech.edu.cn

Weichao Li*
Southern University of Science and Technology, P. R. China
Peng Cheng Laboratory, P. R. China
liwc@sustech.edu.cn

Daoyuan Wu
Singapore Management University, Singapore
dywu.2015@smu.edu.sg

Bo Jin
Southern University of Science and Technology, P. R. China
Peng Cheng Laboratory, P. R. China
jinb@sustech.edu.cn

Rocky K. C. Chang
The Hong Kong Polytechnic University, Hong Kong
csrchang@comp.polyu.edu.hk

Debin Gao
Singapore Management University, Singapore
dbgao@smu.edu.sg

Yi Wang
Southern University of Science and Technology, P. R. China
Peng Cheng Laboratory, P. R. China
wy@ieee.org

Ricky K. P. Mok
University of California, San Diego & CAIDA, USA
cskpmok@caida.org

ABSTRACT

Monitoring mobile network performance is critical for optimizing the QoE of mobile apps. Until now, few studies have considered the actual network performance that mobile apps experience in a per-app or per-server granularity. In this paper, we analyze a two-year-long dataset collected by a crowdsourcing per-app measurement tool to gain new insights into mobile network behavior and application performance. We observe that only a small portion of WiFi networks can work in high-speed mode, and more than one-third of the observed ISPs still have not deployed 4G networks. For cellular networks, the DNS settings on smartphones can have a significant impact on mobile app network performance. Moreover, we notice that instant messaging (IM) and voice over IP (VoIP) services nowadays are not as performant as Web services, because the traffic using XMPP experiences longer latencies than HTTPS. We propose an automatic performance degradation detection and localization method for finding possible network problems in our huge, imbalanced and sparse dataset. Our evaluation and case studies show that our method is effective and the running time is acceptable.

ACM Reference Format:
Shiwei Zhang, Weichao Li, Daoyuan Wu, Bo Jin, Rocky K. C. Chang, Debin Gao, Yi Wang, and Ricky K. P. Mok. 2019. An Empirical Study of Mobile Network Behavior and Application Performance in the Wild. In *IEEE/ACM International Symposium on Quality of Service (IWQoS '19), June 24–25, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3326285.3329039

*Weichao Li is the corresponding author.

1 INTRODUCTION

Measuring and understanding mobile users' received network performance is an essential step towards improving their quality of experience (QoE). Previous research often adopted controlled testbeds or experiments, which allow them to make comprehensive and low-level observations, e.g., down to the radio layer [21, 32, 33, 52]. These studies conducted in-depth performance analysis about WiFi [22, 40, 50], cellular [16, 27, 30], and specific scenarios [28, 35, 39]. However, such controlled approaches are generally not scalable to real users due to the difficulty of deploying them on customized or rooted devices.

To capture real users' QoE, recent works have leveraged crowdsourcing by deploying measurement apps to end users' phones. For example, Huang et al. designed two apps to understand the performance of 3G and 4G networks [26, 29], respectively. Sommers and Barford [48] analyzed crowdsourced data from `speedtest.net` to compare the WiFi and cellular performance. More recently, Mobilyzer [38] was proposed to further speed up the app development and improve their capabilities, and Netalyzr [54] was used to characterize middlebox behavior in cellular networks. However, due to the restriction of collecting other on-device apps' network traces, these Speedtest-like crowdsourcing measurements can analyze only the end-to-end performance between their deployed apps and predefined or user-specified servers.

In this paper, we aim to perform an empirical study of real users' mobile network and application performance by utilizing a unique crowdsourcing dataset.[1] This dataset was collected via MopEye [57], an Android app for monitoring per-app network performance

[1] Please refer to https://github.com/daoyuan14/mopeyeDataset.

without root privileges. More specifically, MopEye leverages the `VpnService` API [10] to capture traffic initiated by all other on-device apps, and measure their performance between the smartphone and the app servers.

In summary, we make the following two contributions:

- We empirically analyze MopEye's two-year-long dataset, which comprises a large number of measurements (i.e., 20 million records from 11,200 users in 173 countries). Compared to previous measurement studies [14, 22, 27], our dataset is superior in scale (653 ISPs in 173 countries vs. several ISPs in one country), timespan (23 months vs. dozens of hours or days), and diversity (1,615 vs. a few smartphone models).
- We propose an automatic method to detect and localize possible performance degradations from massive measurement data, which is very useful for both app developers and network administrators to quickly identify events that could hurt users' QoE. However, this is difficult because the measurements collected by MopEye are individual snapshots of end-to-end network status, we have to handle the potential data imbalance and sparsity. To overcome these challenges, we modify the Apriori algorithm [13], a common method in association rule mining, to detect network anomalies by setting up restrictions on support requirements. After that, we perform post-processing to identify the most likely factors that degrade performance. Our evaluation and case studies show that our method is effective and the running time is acceptable.

Besides our methodological contributions in processing a large-scale crowdsourced dataset and detecting performance degradations, we also make the following insightful observations about mobile network usage and app behaviors:

- Overall, WiFi networks still outperform cellular networks in terms of shorter network latency. Although it is claimed that IEEE 802.11ac equipments have become mainstream in the market [11], only a small fraction (6%) of observed networks exceeded the PHY rates of 300Mbps. For cellular networks, 4G technology dominated the MopEye measurements. However, although 5G technology is getting closer to deployment, more than one-third of the observed ISPs still have not deployed 4G networks, most of which were countries in Africa and Asia.
- Our DNS analysis showed that using external DNS resolvers can result in poorer mobile app performance, especially when the resolvers and end users are not located in the same country.
- Among the top four popular protocols seen by MopEye (HTTPS, HTTP, DNS, and XMPP), a common protocol for IM and VoIP applications [25]), XMPP performed the worst. On the other hand, HTTPS exhibited shorter network latency compared to HTTP. For the app server deployment, we found that advertisement servers usually had longer latency and impaired performance if they fully loaded the advertisements.
- IP anycast does not always have positive effect on reducing network latency. But our study shows that service providers can still benefit from it even if DNS-based redirecting is enabled.

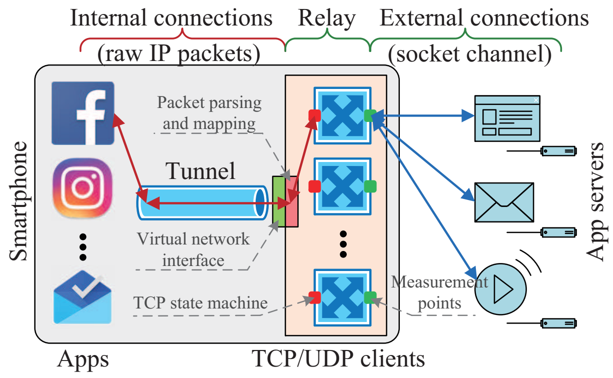


Figure 1: An overview of MopEye.

In the rest of this paper, we first compare traditional smartphone-based crowdsourcing measurement with MopEye in §2. We describe our dataset in §3, and report its new findings on mobile network behavior and application performance in §4. We propose a performance anomaly detection method and evaluate it in §5. We summarize related works in §6 and conclude the paper in §7.

2 BACKGROUND

Smartphone-based crowdsourcing measurement apps (e.g., Ookla Speedtest app[7], MobiPerf [5], and Netalyzr [6]) have attracted a large number of users. Notably, the Ookla Speedtest app[7] has recorded over 100 million downloads on Google Play. These apps often employ the landline measurement paradigm, i.e., measuring network paths only to fixed measurement servers or user-specified remote endpoints. For example, Ookla deployed 7,000+ measurement servers worldwide MobiPerf relied on the M-Lab platform (200+ servers) and server IP addresses specified by the users. Hence, the measurement results they collected correspond to network performance only toward certain (mostly measurement) servers, rather than actual network performance perceived by users.

Recently, some VPN-based measurement apps (e.g., MopEye [57], Haystack [41], and AntMonitor [31]) exploited the VpnService API available on Android 4.0 to enable *in-situ* mobile network measurement, rather than carrying out active measurements to a dedicated set of destinations. MopEye is the first and still the only one of these apps to provide network performance data, while others only collect data for privacy leakage detection. Therefore, in this paper, we focus on MopEye as a representative VPN-based measurement apps. As illustrated in Fig. 1, MopEye maintains and passively monitors a VPN tunnel via the `VpnService` API. Once such a VPN tunnel is established, other apps' traffic will be redirected by the OS to MopEye. Hence, for each network connection, MopEye can record the corresponding app package name, the destination IP address, and the port number. It also acts as a relay to forward network traffic between user apps and the Internet.

By applying typical passive measurement techniques, such as computing TCP 3-way handshake times, and DNS response times, MopEye can estimate the round-trip time (RTT) for external network paths between the smartphone and remote servers for each user app.

Such passive measurements better reflect the actual network performance experienced by the users. For more details about MopEye, please refer to [57].

3 DATASET

In this paper, we analyze the data collected by MopEye from June 1 2016 to May 31 2018[2].

3.1 Data features

Table 1 summarizes the information that MopEye collects, and we call each attribute name a *feature*. User information is collected when a user downloads and installs MopEye. The network information are recorded every time MopEye is enabled or the network state is changed. For every target app server, MopEye will measure it once. More details about the features are described in [57].

Table 1: The features collected in the MopEye measurements.

Feature	Information Group	Description
`DeviceId`	User Information	The unique ID for the device, generated by Android
`UserId`		The unique ID for the user, generated by backend server
`CountrySim`		The registration country of the SIM card
`Device`		The device model information
`AndVer`		The version number of Android
`Type`	Network Information	The network type of the access network connection (e.g., WiFi or cellular network types)
`Name`		The name of the cellular access point, or the SSID of WiFi network
`Longitude` & `Latitude`		The geo-location information of device
`Speed`		The link speed of WiFi network
`Time`		The current timestamp when obtaining the network status
`Rtt`	Measurement and App Information	The RTT of the measured network path
`DestIP`		IP address of the server visited by the app
`DestPort`		Port number of the server visited by the app
`PkgName`		The package name of the measured app
`Detail`		The domain accessed by the app
`Signal`		The signal strength that the smartphone experienced when performing the measurement

3.2 Dataset statistics

Country distribution. The dataset involves more than 11,200 users from 173 countries worldwide. The top 15 user countries include the United States (3,407 users), Indonesia (1,361 users), India (465 users), Malaysia (418 users), and the United Kingdom (407 users), accounting for nearly three quarters of the total users. Besides the user countries, MopEye also collects the geographical locations where measurements are conducted. The geolocation information shows that the MopEye measurements are performed across large populated areas, notably North America, Europe, and Southeast Asia.

Device details. In total, MopEye identified 1,615 different smartphone models from 226 manufacturers. According to the dataset, the top 5 manufacturers (Samsung, LGE, Xiaomi, Huawei, and Motorola) comprised around two-thirds of the total devices. Samsung SM-G935F, Xiaomi Redmi Note 3, and Samsung SM-G930F were the most used smartphone models, which had 205, 153, and 139 users, respectively. We also observed significant Android OS fragmentation in the dataset. The percentages for Android 4, 5, 6, and 7 were 18.55%, 22.64%, 38.03%, and 20.04%, respectively. We only found 95 Android 8 devices.

Applications measured. The dataset includes measurements of 17,059 apps. Among them, 1,197 apps contributed at least 1k measurements, and Facebook (`com.facebook.katana`) the most measured app, was measured 575,529 times. For the top 19 apps that had more than 100k measurements, eight of them were developed by Google and three by Facebook. Other popular apps included system apps such as Clean Master and ES File Explorer, and communication or social networking apps such as TextNow, Telegram, WhatsApp, and WeChat.

Measurements collected. The dataset contains 19,694,295 RTT measurements. 13,204,649 of which are for TCP connections used by the apps, and the remaining 6,489,646 are for DNS measurements. Altogether the dataset covers 286,404 destination IP addresses, 90,902 destination server domains, 8,114 destination server ports, and 3,106 DNS servers. The most accessed domain is `graph.facebook.com`, with 533,698 connections (2.7% of all connections).

Network types. 65.42% of the measurements were performed from WiFi networks. Although MopEye cannot operate the IEEE 802.11 protocols through system APIs directly, it does collect the link speed of WiFi networks. The dataset shows that only a very small portion (5.94%) of measurements exceeded PHY rates of 300Mbps, and the highest speed that we observed was 1,083Mbps. Low-speed WiFi networks ($\leq$54Mbps) comprise more than 30% of measurements. For cellular network measurements, a total of ten network types were recorded. In our analysis, LTE dominated MopEye's cellular measurements: over 70% of the records. However, among the 653 ISPs identified in the dataset, more than one-third of the ISPs (238 ISPs) had no 4G measurements observed. These ISPs belong to 39 countries, mainly in Africa and Asia.

3.3 Network performance overview

We first summarize the overall network performance observed. Our analysis shows that 42.5% of the RTTs were below 50ms and 67% were below 100ms. Moreover, 18.4% of the measurements suffered from relatively long RTTs (>200ms). Expectedly, WiFi showed much better performance than that in cellular networks, especially when network RTTs were short. For example, only a few RTT samples (2.4%) were smaller than 20ms for cellular access, but around 20% for WiFi. The median RTTs for WiFi and cellular networks were 53ms and 75ms, respectively.

To better understand the QoS of current mobile networks, we further calculated the median RTT and latency jitters of LTE networks for major ISPs. We defined jitter as the median standard deviation of the RTTs to the same destination IP in a measurement session. Considering the fact that local DNS resolvers of cellular carriers are very close to end users, with relatively short network latency [45] and few content providers can deploy their services closer than that, we calculate these metrics using DNS measurements to estimate the performance upper bound that a cloud service provider might achieve. The results of the top 15 ISPs observed is plotted in Fig 2.

[2]It is the latest dataset MopEye authors provided at the time of our study.

We can see that all ISPs can provide decent network latency even for applications that have high requirements. For example, online gamers often expect latency less than 100ms [17]. However, many ISPs experienced high jitter, suggesting that users of cloud gaming or VR streaming applications may still experience lags using today's 4G network.

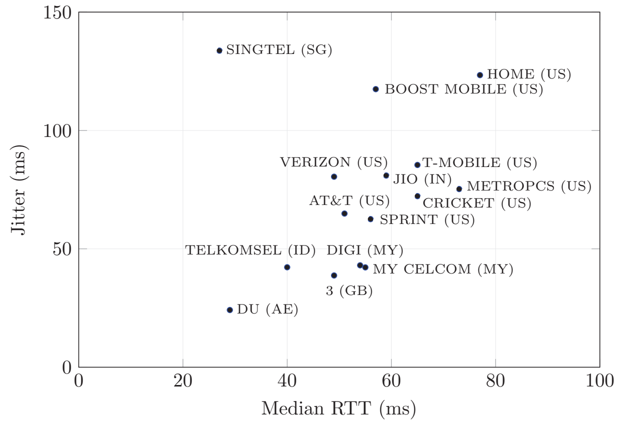


Figure 2: Median latency and network jitters of major ISPs.

4 MOBILE APPLICATIONS' NETWORK PERFORMANCE

4.1 Protocols

We next study the network performance for different protocols, which can be identified through feature `DestPort`. Based on our analysis, the most used destination ports are 443 and 80, corresponding to HTTPS and HTTP, respectively. Port 53 (DNS) ranks next, followed by ports 5222, 5223 and 5224 (used by XMPP). We plot the distribution of their RTTs in Fig. 3 for these popular ports. As expected, port 53 outperforms other three, because DNS resolvers are usually placed as close as possible to end users. Moreover, port 443 performs better than port 80. A reasonable explanation is that since modern websites have already migrated their major services to HTTPS [36], HTTPS may receive more optimization in both web servers and network policies. The XMPP ports, however, have much longer network latency. For example, more than a half of the measured RTTs are longer than 133ms. Implying much room for improvement in today's IM and VoIP services.

4.2 DNS performance

DNS plays a critical role in the performance of mobile applications for name resolving, directing users to the closest cache servers and performing load-balancing [19, 47, 58]. We are interested in how the deployment of DNS servers affects the cellular network performance experienced by users. Generally, most cellular ISPs provide DNS services to their customers, which are usually deployed very close to end users with private IP addresses. Besides, there are some public DNS services such as Google Public DNS and OpenDNS, which are hosted outside the ISPs and configured with public IP addresses.

By querying for geolocation and ISP information through on-line services such as Geo IP Lookup [3], we obtained countries and ISP

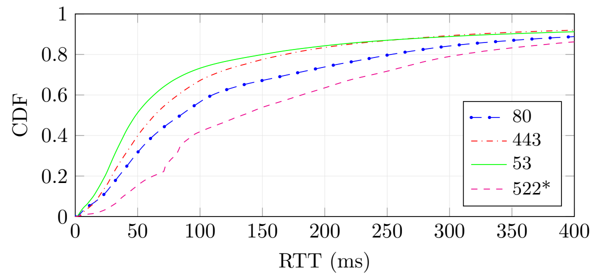


Figure 3: CDF plot of median RTT for different ports. "522*" includes 5222, 5223, and 5224.

names of those DNS server IP addresses. We determined whether they belonged to the same ISP and country as the mobile end users. Accordingly, we classified the DNS server IP addresses into four groups: i) private IPs; ii) public IPs owned by the same ISP; iii) public IPs within the same country but owned by other ISPs; and iv) public IPs in a different country. Here, the former two groups could be local DNS servers while the latter two could be external ones. Since we do not consider IP anycast in this section, we put measurements using public DNS with IP anycast support into group iv. In our data, the four groups contained 25%, 24%, 45%, and 6% measurements, respectively.

Fig. 4 plots the distribution of DNS resolving time for the four groups of DNS deployments. As shown, when the DNS servers are located in different countries, users always experienceed higher DNS query delays. For the other three groups of DNS servers within the same country, they performed similarly, regardless of whether they were hosted in the same ISP. Fig. 5 plots the CDF of network delays experienced by mobile apps when using different types of DNS servers. Significantly, mobile apps may experience poorer network quality if DNS servers in other countries are used. For the other three groups of DNS servers, mobile apps can experience comparably short network latencies. More specifically, apps with local private DNS servers performed slightly better than those cases where DNS servers were hosted in the same ISP, and the same country. We conclude that using third-party DNS resolvers (such as public DNS services) may not improve mobile network performance, but could result in deterioration if resolvers are not close to the end users.

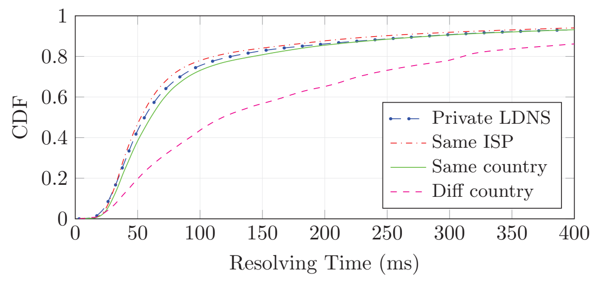


Figure 4: CDF plot of DNS resolution time for different DNS deployments. The time includes both network delay and server response time (e.g., its recursive querying time).

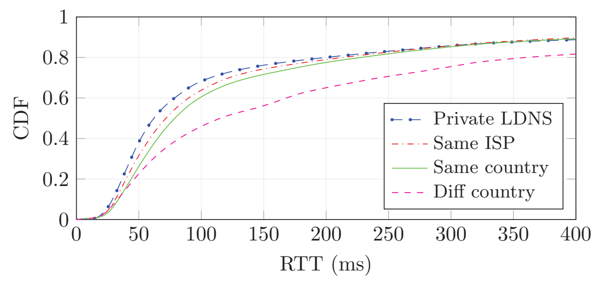
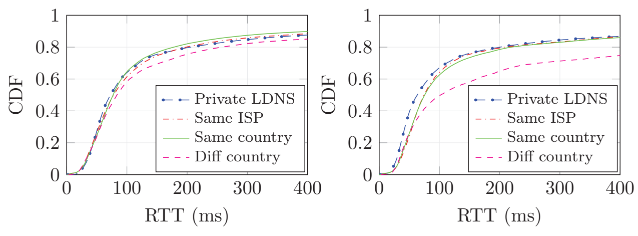


Figure 5: CDF plot of RTTs to app servers for different DNS deployments.

To validate whether this conclusion is still reasonable for popular apps, we analyzed the network performance of Facebook and YouTube. As illustrated in Fig. 6(a) and 6(b), although Facebook performs similar when adopting different types of DNS resolvers, both apps had shorter RTTs when using local private DNS resolvers and longer RTTs for the cases in different countries. Such large content providers may have already optimized their server deployment well enough, and end users are unlikely to improve performance by switching to external DNS servers.



(a) Facebook. (b) YouTube.

Figure 6: CDF plots of RTTs for Facebook and YouTube when using different types of DNS resolvers.

4.3 Application servers

In order to serve more users and provide better performance, modern mobile apps are often served by hundreds of servers, which is confirmed in our dataset. As demonstrated in Fig. 7, four of the most popular apps (Facebook, YouTube, CleanMaster, and TextNow) were observed with more than 10,000 unique IP addresses for the destination app servers. Those servers could be owned by the app service providers themselves, third-party CDN servers (e.g., `cloudfront.net` and `cloudflare.com`), advertisement servers (e.g., `adnxs.com` and `googlesyndication.com`), trackers including analytic and statistic services (e.g., `quantserve.com` and `crashlytics.com`), and other servers that could not be identified. We used EasyList [2] to identify advertisement and tracker domains.

Table 2 summarizes the percentage of measurements and median RTTs of each type of server for the four aforementioned apps. For Facebook and YouTube, the majority of network access went to their own servers. But for CleanMaster and TextNow, the most visited servers were advertisement servers, accounting for 58.1% and 54.7%, respectively. Moreover, advertisement servers usually performed worse than their own servers for all four apps. This means that improper design of apps (e.g., rendering GUI after fully loading the advertisements) could prevent the apps from responding quickly, resulting in QoE degradation.

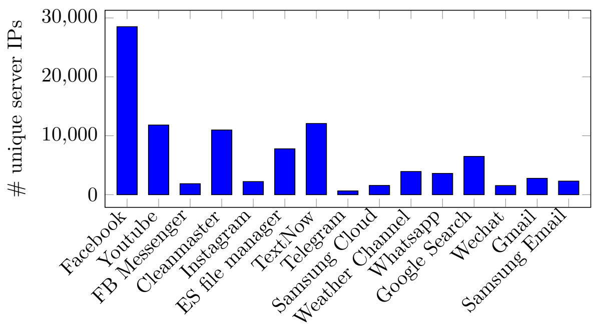


Figure 7: Number of unique server IP addresses for the top 15 apps with most measurement records.

Table 2: The roles, percentage of measurements, and median RTTs of different server types for the four top apps.

App	Server role	Percentage	Median RTT (ms)
Facebook	Own server	53.3%	54
	3rd-party CDN	2.2%	86
	Ad	9.6%	71
	Tracker	3.1%	71
YouTube	Own server	81.5%	49
	3rd-party CDN	0%	n/a
	Ad	1.1%	56
	Tracker	0.9%	100
CleanMaster	Own server	13.0%	96
	3rd-party CDN	0.9%	50
	Ad	58.1%	96
	Tracker	4.2%	79
TextNow	Own server	4.0%	66
	3rd-party CDN	2.6%	80
	Ad	54.7%	73
	Tracker	10.4%	61

4.4 IP anycast deployment and performance

Many mobile applications employ geographically distributed content delivery networks (CDNs) to reduce the latency for users to access content. Assigning users to the nearest cache/replica is a key factor in improving the CDN performance. Current approaches for server selection include HTTP redirection, URL rewriting, anycast, and DNS-based server redirecting [55]. We study the deployment and performance of anycast servers used by mobile apps.

We first matched the destination IP addresses in the dataset with the anycast IP addresses listed in the "2017-04" data from [1], which was generated by the iGreedy algorithm [23]. We found that about 8.32% of the measurements (2.29% of all destination IP addresses) connect to anycast servers. To fairly compare the performance between anycast and unicast, we considered the IP addresses belonging to the same second-level domain (SLD), because they are likely under the control of the same company. We show the results of some typical SLDs in Table 3. In the table, the columns #Anycast and

#Unicast represent the numbers of RTT measurements in which the destination IP address was classified as anycast and unicast, respectively. R_u, R_a, and ΔR represent the median RTT of unicast, anycast IP addresses, and their difference ($R_a - R_u$), respectively.

Table 3: The deployment and performance of IP anycast in several SLDs.

SLD	#Unicast	#Anycast	R_u (ms)	R_a (ms)	ΔR (ms)
reddit.com	457	236	34	44	10
pinterest.com	5486	1079	61	39	-22
cloudflare.com	0	1629	n/a	47	n/a
googleapis.com	745881	0	45	n/a	n/a

We found that some SLD anycast IP addresses have higher RTT than unicast. One possible explanation is that anycast packets suffer from some routing problems and result in suboptimal replicas [56], while DNS-based redirection may already direct the user to the nearest server. For validation, we employed the average number of IP addresses for each domain and a robust regression method. Here the number of IP addresses we treated as a measure of DNS-based redirection utilization, and used robust regression to identify a correlation between the performance gain from anycast and the utilization of DNS-based redirection. We used `rlm()` from R package `MASS` with default parameters to perform robust regression, which internally uses the iteratively re-weighted least squares (IRLS) algorithm [20] to build a linear model that assigns less weight to outlier data to minimize their impact. We plot the 22 SLDs that had more than 10 anycast IP addresses in our dataset in Figure 8.

The plot shows a relatively strong positive correlation, suggesting that anycast can still significantly reduce latency to application servers even when DNS-based redirection is deployed. It is reasonable because many mobile users do not use their local DNS server (as discussed in § 4.2), and thus the DNS-based redirection could fail but IP anycast remains effective.

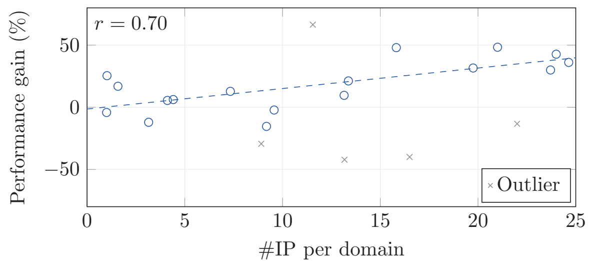


Figure 8: Relationship between performance gain by using IP anycast and the number of IP per domain. Each point is an SLD. Outliers are identified by robust regression. r stands for the Pearson correlation coefficient.

5 PERFORMANCE DEGRADATION DETECTION

5.1 Challenges

So far, we have analyzed our dataset empirically and highlighted our findings manually. However, it is a time-consuming task that could miss a lot of meaningful events. Therefore, we need a method to automatically detect and localize possible performance degradations. There are several challenges for this task. We summarize the challenges as follows.

First of all, our dataset is highly imbalanced. Some users may produce thousands of measurements, but others only contribute a dozen. Owing to that, it is difficult to draw an unbiased conclusion based on standard frequent item mining methods that focus only on the number of records, because the majority of records may come from a single user. A typical example is that 83.5% of the 16,868 HSPAP measurements for ISP Mobilis are from one user. If those measurements are excluded, the median RTT decreases from 332ms to 219ms.

The second challenge is the sparsity of our dataset. Although a huge number of measurements and plenty of network information were collected by MopEye, they are individual snapshots of end-to-end network performance. We may not have enough observations for some combinations of features. For example, small carriers and uncommon devices usually generate very few data samples. Therefore we may not have enough historical records for time-series analysis.

The last challenge arises from the data volume. With the growth of MopEye users, it is inevitable that more measurements are conducted every day (e.g., MopEye recorded ~130K measurements in June 2016 when MopEye was just released, and the number increased to ~300K in April 2018). Accordingly, we need a scalable algorithm to efficiently process the growing dataset.

5.2 Our method

We first introduce the notation used in our proposed method. As discussed in §3.1, MopEye collects a set of features for each measurement record. We define a *property* as a feature-value pair. For example, a record with property "date:201706" means that it was measured in June 2017. Similarly, a *condition* is defined as a group of properties from different features. We use $f(y)$ to express the feature that a property y belongs to, and similarly $F(\boldsymbol{x})$ to express the set of features that the properties in condition $\boldsymbol{x}$ belong to. Note that no two properties of a condition belong to the same feature, i.e., $|\boldsymbol{x}| = |F(\boldsymbol{x})|$ for any condition $\boldsymbol{x}$.

Let D be the whole dataset. For a condition $\boldsymbol{x}$, $D_{\boldsymbol{x}}$ represents the records that match all properties in $\boldsymbol{x}$, and $R_{\boldsymbol{x}}$ represents the corresponding RTTs. Table 4 summarizes the notation used in this section.

Table 4: Summary of notations.

Notation	Description
$f(y)$	The feature of a property y
$F(\boldsymbol{x})$	The feature set of properties in a condition $\boldsymbol{x}$
$D_{\boldsymbol{x}}$	The records that match all properties in a condition $\boldsymbol{x}$
$R_{\boldsymbol{x}}$	The vector of RTTs in $D_{\boldsymbol{x}}$
$\lvert\cdot\rvert$	The cardinality of a set or a vector

We propose a method based on the association rule mining techniques. An association rule of the form "$\boldsymbol{x}_a \Rightarrow \boldsymbol{x}_c$" means if a record contains all properties in $\boldsymbol{x}_a$, it is likely to contain $\boldsymbol{x}_c$. In practice, we often restrict the properties that can be included in $\boldsymbol{x}_c$. In our tasks, we have $\boldsymbol{x}_c$ represent the degradation or non-degradation event, and $\boldsymbol{x}_a$ represent the conditions that lead to the degradation.

However, since our dataset is collected worldwide and covers various kinds of networks, it is hard to label a record as degradation or non-degradation. For example, a network latency of 200ms is usually undesirable, but it is expected for 2G users or for users who access websites located on a different continent.

Therefore instead of using pre-defined labels, we compare the average performance of different properties under certain conditions to find anomaly rules. We define *performance degradation event* as follows:

DEFINITION 1. *For a condition $\boldsymbol{x}$ and a property y such that $f(y) \notin F(\boldsymbol{x})$, if the* $\text{median}(R_{\boldsymbol{x}\cup\{y\}})$ *is significantly higher than* $\text{median}(R_{\boldsymbol{x}})$ *with enough support (§ 5.2.1), we say $(\boldsymbol{x}, y)$ is a performance degradation event.*

We employ the median absolute deviation (MAD), denoted by $\text{MAD}(R_{\boldsymbol{x}})$, as a measure of significance (the definition of support is discussed in § 5.2.1). The goal of our method can be summarized as: finding all possible degradations that meet the above criteria, and automatically localizing the root cause. We modify the association rule mining techniques to achieve the goal. Our method consists of three steps. First, we find frequent items as in association rule mining. Then, we detect performance degradations among frequent items according to our definition of performance degradation. Finally, we examine other possible root causes for each performance degradation event to localize the root cause.

5.2.1 Frequent item mining.

Frequent items are conditions that meet some specific support requirements. Usually the support is defined by a minimum requirement of the number of records, denoted by n. However, the imbalanced nature of our data can tend to biased conclusions, e.g., if most records come from a single user. So in addition to a minimum number of records, we also require that the unique number of users in $D_{\boldsymbol{x}}$ of a condition $\boldsymbol{x}$ is no less than 5, and no single user takes up more than 50% of the records. We choose $n = 750$ in our method. We give a rationale for choosing this value in §5.4.1.

We use a modified Apriori algorithm to collect frequent items. The original Apriori [13] is a popular algorithm which can find all property combinations in a dataset that appear at least n times. Since a minimum number of records is not sufficient to support our rules, we add our requirements into the algorithm. We start with an initial candidate set $C^0 = \{\emptyset\}$. In each iteration, we generate candidate conditions

$$C^k = \left\{ \boldsymbol{x} \cup \{y\} \;\middle|\; \begin{array}{l} \boldsymbol{x} \in C^{k-1}, \\ f(y) \notin F(\boldsymbol{x}), \\ |D_{\boldsymbol{x}\cup\{y\}}| \geqslant n, \\ U_{\boldsymbol{x}\cup\{y\}} \geqslant 5 \end{array} \right\}$$

by looping over all combinations of $\boldsymbol{x} \in C^{k-1}$ and property y such that $f(y) \notin F(\boldsymbol{x})$, to prune the candidates that do not meet the support requirement. $U_{\boldsymbol{x}}$ in the equation stands for the number of unique users in $D_{\boldsymbol{x}}$. The iteration ends when $C^k = \emptyset$. Finally, we filter all candidates in $\bigcup_{i=0}^{k} C^i$ by the last requirement that no single user shall take up more than 50% of the records.

All frequent items can be found in this way because we have the downward closure property [12], i.e., $|D_{\boldsymbol{x}}| \geqslant |D_{\boldsymbol{x}\cup\{y\}}|$ and $U_{\boldsymbol{x}} \geqslant U_{\boldsymbol{x}\cup\{y\}}$ for any $\boldsymbol{x}$ and y. For the remaining requirement that no single user shall take up more than 50% of the records, we cannot add it in the pruning step since the downward closure property does not hold for it. We thus drop the frequent items that do not meet this requirement after all frequent items are mined.

5.2.2 Performance degradation detection.

Recalling Definition 1, a performance degradation event is a frequent item $\boldsymbol{x}$ and a property y, such that $f(y) \notin F(\boldsymbol{x})$, $\boldsymbol{x} \cup \{y\}$ is also a frequent item, and $\text{median}(R_{\boldsymbol{x}\cup\{y\}}) > \text{median}(R_{\boldsymbol{x}}) + \text{MAD}(R_{\boldsymbol{x}})$. To find all performance degradation events in our data, we loop over condition $\boldsymbol{x}$ in frequent items and calculate $u_{\boldsymbol{x}} = \text{median}(R_{\boldsymbol{x}}) + \text{MAD}(R_{\boldsymbol{x}})$ as the upper bound of normal performance under that condition. We then find all properties y such that $f(y) \notin F(\boldsymbol{x})$, and $\boldsymbol{x} \cup \{y\}$ is also a frequent item, and $\text{median}(R_{\boldsymbol{x}\cup\{y\}}) > u_{\boldsymbol{x}}$. As a result, such $(\boldsymbol{x}, y)$ pairs are therefore performance degradation events by definition.

5.2.3 Root cause identification.

Due to the imbalanced nature of our crowdsourcing data, the property y in a detected performance degradation event may not be the real cause of the performance degradation event. For example, in our dataset, $\text{median}(R_{\text{type:LTE,kernel:3.10.49}}) = 340$, while for the overall RTT of all LTE records, $\text{median}(R_{\text{type:LTE}}) = 73$, and $\text{MAD}(R_{\text{type:LTE}}) = 36$. However, after inspecting the supporting data $D_{\text{type:LTE,kernel:3.10.49}}$, we found that 81.7% records also have "name:JIO4G", while in the complete dataset only 0.72% records have that property. We cannot determine without further analysis if the root cause resides in the Linux kernel version or in the carrier.

To help identify the root cause, the third step of our method analyzes all possible causes of a performance degradation event. We check every possible cause using criteria and hypothesis tests. First, we calculate the marginal frequency $P_z = |D_z| \;/\; |D|$ for every property z. Then we group performance degradation events by their supporting data $D_{\boldsymbol{x}\cup\{y\}}$. For each group and property z, $f(z) \notin F(\boldsymbol{x} \cup \{y\})$, we calculate *confidence* $= P_{z|\boldsymbol{x}\cup\{y\}} = |D_{\boldsymbol{x}\cup\{y,z\}}| \;/\; |D_{\boldsymbol{x}\cup\{y\}}|$ and *lift* = *confidence* / P_z. If both the confidence and lift are high for a property z, it may be another cause for the high latency of the supporting data, because it appears in the supporting data more frequently than elsewhere. In our settings, we require that *confidence* $\geqslant 0.2$ and *lift* $\geqslant 3$. We then use Mann–Whitney U test [37] to check if that property has a significant impact on the RTT of the supporting data. For each property z with both high confidence and lift, we partition the supporting data $D_{\boldsymbol{x}\cup\{y\}}$ to $D_{\boldsymbol{x}\cup\{y,z\}}$ and $D_{\boldsymbol{x}\cup\{y\}} \setminus D_{\boldsymbol{x}\cup\{y,z\}}$ and perform a Mann–Whitney U test on the two subsets. The higher the p-value of the test, the less likely c is a root cause of the performance degradation event. Finally, we sort the performance degradation events by the minimum p-value of all z that could possibly be another root cause of the high RTT of their supporting data, so we can focus on the performance degradation events that can be more independently explained.

5.3 Implementation details

5.3.1 Data preprocessing.

A requirement for frequent items mining techniques is that all items need to be discrete, so we need to preprocess our data. For continuous features, we classify them into several categories based on our knowledge and their distributions. Specifically, we split WiFi link speed by 54Mbps and 300Mbps based on the evolving stages of WiFi standards and split signal strength using the quartiles, which are -72dbm, -56dbm and -39dbm.

We also derive some features from the raw data to find the root cause for performance degradation events in different aspects. For example, we find the nearest city to the user's location using the data from GeoNames [9]. The full list of derived features is shown in Table 5.

Table 5: Derived features.

Feature	Derived from	Description
`City`	`Latitude` & `Longitude`	Nearest city of user location
`DestCountry`	`DestIP`	Registration country of the DestIP
`DestISP`	`DestIP`	Registration ISP of the DestIP
`Brand`	`Device`	The manufacturer of the device
`Pkg2`	`PkgName`	First 2 segments of the package name (usually the company)
`Pkg3`	`PkgName`	First 3 segments of the package name (usually the product series)
`AndVer2`	`AndVer`	The minor (first two parts) Android version of user equipment
`Kernel2`	`Kernel`	The minor (first two parts) Linux kernel version of user equipment
`Date2`	`Time`	The date rounded to month (yyyy-mm) when obtaining the network status
`Date3`	`Time`	The date rounded to day (yyyy-mm-dd) when obtaining the network status

5.3.2 Distributed computing. Our dataset is large, and our method needs to find all possible performance degradation events and examine all possible causes for each performance degradation event. Thankfully, the algorithm we described in §5.2 can run in parallel. We use the built-in parallel computing feature of Julia, the language in which we implement our method, to run our system in a cluster. We store our data in an SQLite database. Every computer in the cluster has a full copy of all data. We organize our computation tasks by processes. Different processes may or may not run on the same computer. There are two types of processes: a scheduler and several workers. The scheduler maintains a task queue, and distribute tasks to and gathers results from workers. The workers connect to the scheduler over networks and perform the actual computations.

The first step is frequent item mining. We run our modified Apriori algorithm in parallel as follows. The scheduler starts with a candidate set that contains only an empty set. For each element $\boldsymbol{x}$ in the candidate set, the scheduler generates y so that $f(y) \notin F(\boldsymbol{x})$, and distributes this y to an idle worker. The worker scans the database and reports if $\boldsymbol{x} \cup \{y\}$ meets the requirement or not. If $\boldsymbol{x} \cup \{y\}$ meets the support requirement, the scheduler then adds it to the candidate set and generates new tasks based on it.

Several optimizations can be applied in scheduling tasks. Notably in the second step, a worker examines if a property y can increase the median RTT of $D_{\boldsymbol{x}}$ by $\mathrm{MAD}(R_{\boldsymbol{x}})$. In our implementation, each worker slices the relevant data $D_{\boldsymbol{x}}$ and loads it into memory at the beginning of a task. The workers also remember their previous tasks $(\boldsymbol{x}, y)$, and if the new task with condition $\boldsymbol{x}'$ has $\boldsymbol{x}' \subseteq \boldsymbol{x}$, it will simply slice the previously loaded data, instead of reading it from the disk. This optimization technique can saves a large number of I/O operations.

5.4 Evaluation

5.4.1 Mathematical analysis. We first verify that our definition of performance degradation is unlikely to cause false positives in a simplified model. In the model there are k ($k \geqslant 2$) properties of a feature Y and each property has n independent records ($n \geqslant 750$). We use $y_1, \ldots, y_k$ to denote the properties of feature Y. We prove that if there is no anomaly, i.e., all RTTs are subject to the same distribution $\mathcal{D}$ independently, then we are unlikely to have a property y ($f(y) = Y$) such that y meets our definition of performance degradation event in the beginning of §5.2. We omit other conditions $\boldsymbol{x}$ in this analysis and focus only on y.

Since we have a relatively large n, according to Laplace's early work [49], the distribution of the median is approximately a normal distribution. i.e., $\mathrm{median}(R_{y_i}) \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu, \sigma^2)$. The μ of $\mathrm{median}(R_{y_i})$ is equal to the median of $\mathcal{D}$. Given that we do not know the actual distribution $\mathcal{D}$ of RTT, we use the empirical distribution $\hat{\mathcal{D}}$ of our over 20 million records as an approximation, which has median = 72 and MAD = 42. It is also possible to estimate σ^2 of $\mathrm{median}(R_y)$ [42], but the formula is complicated and tricky to implement. For example, we need to apply some kind of smoothing to our empirical distribution, to obtain a proper probability density function. For simplicity, we use the sampling method to estimate σ^2 directly. That is, we draw S_i ($1 \leq i \leq N = 100,000, |S| = n$) from the empirical distribution $\hat{\mathcal{D}}$ and calculate the medians $M_i = \mathrm{median}(S_i)$. Then we use the sample variance $\sum_{i=1}^{n}(M_i - \bar{M})^2 \,/\, (N-1)$ as the estimator of σ^2.

We are now interested in the probability of false positives, i.e., the probability that our method thinks that there is an anomaly in arbitrary data. According to the definition, a degradation y is detected if $\mathrm{median}(R_y) \geq \mathrm{median}(R) + \mathrm{MAD}(R)$. In principle $\mathrm{median}(R_y)$, $\mathrm{median}(R)$ and $\mathrm{MAD}(R)$ are not independent. For simplicity, we treat $\mathrm{median}(R)$ and $\mathrm{MAD}(R)$ as constants, denoted by μ and MAD, respectively. Then the probability of flagging anomalies can be calculated as follows:

$$\begin{aligned}
P &= Pr[\ \exists i\ s.t.\ \mathrm{median}(R_{y_i}) \geq \mu + \mathrm{MAD}] \\
&= 1 - \prod_{1 \leq i \leq k} Pr[\mathrm{median}(R_{y_i}) < \mu + \mathrm{MAD}] \\
&= 1 - Pr[\frac{\mathrm{median}(R_{y_1}) - \mu}{\sigma^2} < \frac{\mathrm{MAD}}{\sigma^2}]^k \\
&= 1 - Pr(Z < \frac{\mathrm{MAD}}{\sigma^2})^k \quad (Z \sim \mathcal{N}(0,1)) \\
&= 1 - \Phi(\frac{\mathrm{MAD}}{\sigma^2})^k
\end{aligned}$$

We calculate the probability with different values of k and n using our estimated $\hat{\sigma}^2$. The result is shown in Table 6. As the table shows, the probability is quite small when $n > 750$. Since we have a minimum support requirement of 750 for every performance degradation and many of them have actual support requirements larger than 1500, we argue that our method is unlikely to yield false positives.

Table 6: Probability table of false positives.

	$k = 2$	$k = 5$	$k = 15$
$n = 500$	0.013	0.033	0.095
$n = 750$	$2.02 * 10^{-4}$	$5.05 * 10^{-4}$	$1.51 * 10^{-3}$
$n = 1000$	$8.78 * 10^{-7}$	$2.19 * 10^{-6}$	$6.58 * 10^{-6}$
$n = 1500$	$1.75 * 10^{-13}$	$4.37 * 10^{-13}$	$1.31 * 10^{-12}$

5.4.2 Real-world performance degradation events. We present two interesting cases found by our algorithm.

The first performance degradation event is related to time. Our algorithm found that the RTT of the records whose destination IP address geolocates to Germany and its ISP is Google, was higher than usual around August 2017. It finds significant commonality among the supporting data, suggesting that this performance degradation event was likely not due to the random bias of a few users. We plot the RTTs of the related records as a time series in Fig. 9.

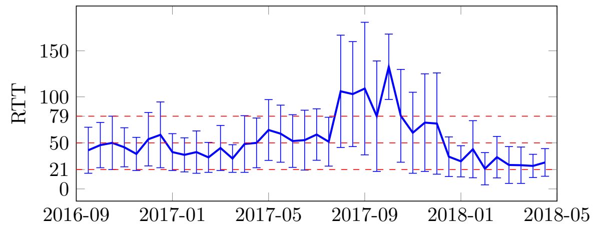


Figure 9: RTT of Google in Germany. Error bars use ±MAD**. The dashed lines are overall** median ± MAD**.**

The second performance degradation event involved servers of Microsoft Office Mobile. In our dataset, there are 96,997 records of 1,624 IPs that were accessed by `com.microsoft.office.*`. Most of the IPs are registered under Microsoft Corporation, Amazon, Akamai Technologies, and Cloudflare. Our algorithm found that in terms of median RTT, servers that have IP registered under Amazon performed much worse than others. We plot the CDF of RTTs of the four ISPs in Fig. 10. We further inspected the supporting data and found that most of the IPs registered under Amazon resolved from the domain `files.acompli.net` and `api.acompli.net`, which were accessed by Outlook.

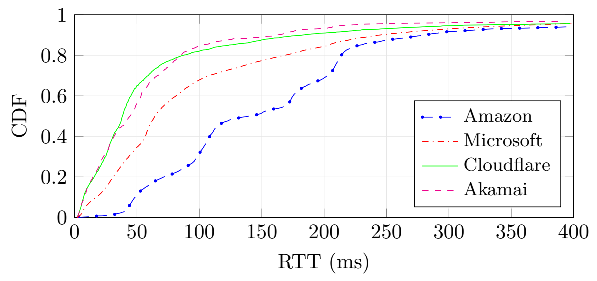


Figure 10: CDF plot of RTTs of different ISPs accessed by `com.microsoft.office.*`.

6 RELATED WORKS

Monitoring mobile network performance through smartphone apps has gained popularity among end users [4, 6, 8]. These apps or platforms can be also applied to tackling various network performance issues [38, 43, 44, 53, 54]. However, they are still based on the landline measurement paradigm, which measures network paths from smartphones to some particular measurement servers or user-specified remote endpoints. Therefore, the measurement results only correspond to the network performance between the users and the target servers, but cannot capture performance that normal apps may perceive. On the other hand, MopEye [57], Haystack [41], and AntMonitor [31] leverage the `VpnService` API [10] to capture the real app traffic, and can measure actual network performance experienced by end users.

Network performance degradation is an important topic for network providers and the methods to detect it have been long studied. However, most of them use data that are continuous and homogeneous. For example, Yan et al. [59], Amrutkar et al. [15] and Ahmed et al. [14] collected data in a continuous time span from the core network of a single ISP. Sun et al. [51] adopted a Monte Carlo Tree Search approach for anomaly localization, specifically for additive key performance indicator (KPI) values.

Internet IP anycast as a widely used technique is also wildly studied. Most studies have focused on root DNS servers and CDNs [18, 34, 46]. Unlike our work, most previous measurements used a limited number of probes, which may not accurately reflect the experience of real users. For example, PlanetLab is used by [24]. However, almost all nodes of PlanetLab are located in universities, which do not represent typical users.

7 CONCLUSION

In this paper, we analyzed a two-year-long dataset collected by MopEye. By inspecting the dataset in detail, we gained a much better understanding of mobile network behavior and application performance. For example, we found that very few WiFi networks can exceed the PHY rates of 300Mbps, and DNS settings can directly affect the mobile apps' network performance in cellular networks. Moreover, we noticed that traffic using the XMPP protocol had higher latency than HTTPS, which implies that there is still room for improvement for today's IM and VoIP services. Besides the in-depth analytic, we also proposed an automatic network performance degradation detection method, which can help to find possible network performance problems in a huge, imbalanced and sparse dataset.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments. This work is partially supported by an NSFC project grant (ref. no. 61872420), and the project of "PCL Future Regional Network Facilities for Large-scale Experiments and Applications (ref. no. PCL2018KP001)".

REFERENCES

[1] Anycast dataset. https://anycast.telecom-paristech.fr/dataset/.
[2] Easy List. https://easylist.to/.
[3] Geo IP Lookup. http://geoiplookup.net/.
[4] MobiPerf. http://www.mobiperf.com/.
[5] MobiPerf on Google Play. https://play.google.com/store/apps/details?id=com.mobiperf.
[6] Netalyzr on Google Play. https://play.google.com/store/apps/details?id=edu.berkeley.icsi.netalyzr.android.
[7] Speedtest by Ookla. https://play.google.com/store/apps/details?id=org.zwanoo.android.speedtest.
[8] Speedtest.net on Google Play. https://play.google.com/store/apps/details?id=org.zwanoo.android.speedtest.
[9] The GeoNames geographical database. http://geonames.org/.
[10] VpnService on Android Developers. http://developer.android.com/reference/android/net/VpnService.html.
[11] Worldwide enterprise WLAN market sees steady growth in full year and Q4 2017, according to IDC. https://www.idc.com/getdoc.jsp?containerId=prUS43599518.

[12] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 207–216. ACM, 1993.

[13] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499. Morgan Kaufmann Publishers Inc., 1994.

[14] F. Ahmed, J. Erman, Z. Ge, A. X. Liu, J. Wang, and H. Yan. Detecting and localizing end-to-end performance degradation for cellular data services based on TCP loss ratio and round trip time. *IEEE/ACM Transactions on Networking*, 25(6):3709–3722, Dec 2017.

[15] 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 *Proc. IEEE/IFIP DSN*, 2013.

[16] D. Baltrunas, A. Elmokashfi, and A. Kvalbein. Measuring the reliability of mobile broadband networks. In *Proc. ACM IMC*, 2014.

[17] W. Cai, R. Shea, C.-Y. Huang, K.-T. Chen, J. Liu, V. C. Leung, and C.-H. Hsu. The future of cloud gaming [point of view]. *Proceedings of the IEEE*, 104(4):687–691, 2016.

[18] M. Calder, A. Flavel, E. Katz-Bassett, R. Mahajan, and J. Padhye. Analyzing the performance of an anycast cdn. In *Proceedings of the 2015 Internet Measurement Conference*, IMC '15, pages 531–537, New York, NY, USA, 2015. ACM.

[19] V. Cardellini, M. Colajanni, and P. S. Yu. Dynamic load balancing on web-server systems. 3(3):28–39, 1999.

[20] R. Chartrand and W. Yin. Iteratively reweighted algorithms for compressive sensing. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3869–3872, March 2008.

[21] Q. A. Chen, H. Luo, S. Rosen, Z. M. Mao, K. Iyer, J. Hui, K. Sontineni, and K. Lau. QoE Doctor: Diagnosing mobile app QoE with automated UI control and cross-layer analysis. In *Proc. ACM IMC*, 2014.

[22] X. Chen, R. Jin, K. Suh, B. Wang, and W. Wei. Network performance of smart mobile handhelds in a university campus WiFi network. In *Proc. ACM IMC*, 2012.

[23] D. Cicalese, J. Augé, D. Joumblatt, T. Friedman, and D. Rossi. Characterizing ipv4 anycast adoption and deployment. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, pages 16:1–16:13, New York, NY, USA, 2015. ACM.

[24] D. Cicalese, J. Augé, D. Joumblatt, T. Friedman, and D. Rossi. Characterizing ipv4 anycast adoption and deployment. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, pages 16:1–16:13, New York, NY, USA, 2015. ACM.

[25] A. Hornsby and R. Walsh. From instant messaging to cloud computing, an XMPP review. In *IEEE International Symposium on Consumer Electronics (ISCE 2010)*, pages 1–6, 2010.

[26] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. A close examination of performance and power characteristics of 4G LTE networks. In *Proc. ACM MobiSys*, 2012.

[27] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck. An in-depth study of LTE: Effect of network protocol and application behavior on performance. In *Proc. ACM SIGCOMM*, 2013.

[28] J. Huang, F. Qian, Z. M. Mao, S. Sen, and O. Spatscheck. Screen-off traffic characterization and optimization in 3G/4G networks. In *Proc. ACM IMC*, 2012.

[29] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing application performance differences on smartphones. In *Proc. ACM MobiSys*, 2010.

[30] H. Jiang, Y. Wang, K. Lee, and I. Rhee. Tackling bufferbloat in 3G/4G networks. In *Proc. ACM IMC*, 2012.

[31] A. Le, J. Varmarken, S. Langhoff, A. Shuba, M. Gjoka, and A. Markopoulou. Antmonitor: A system for monitoring from mobile devices. In *C2BD SIGCOMM*, 2015.

[32] W. Li, R. K. P. Mok, D. Wu, and R. K. C. Chang. On the accuracy of smartphone-based mobile network measurement. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 370–378, April 2015.

[33] W. Li, D. Wu, R. K. Chang, and R. K. Mok. Demystifying and puncturing the inflated delay in smartphone-based wifi network measurement. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, pages 497–504, New York, NY, USA, 2016. ACM.

[34] Z. Li, D. Levin, N. Spring, and B. Bhattacharjee. Internet anycast: Performance, problems, &#38; potential. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 59–73, New York, NY, USA, 2018. ACM.

[35] Q. Liu, K. Xu, H. Wang, M. Shen, L. Li, and Q. Xiao. Measurement, modeling, and analysis of TCP in high-speed mobility scenarios. In *Proc. IEEE ICDCS*, 2016.

[36] G. LLC. HTTPS encryption on the web. https://transparencyreport.google.com/https/.

[37] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. 18(1):50–60, 1947.

[38] A. Nikravesh, H. Yao, S. Xu, D. Choffnes, and Z. M. Mao. Mobilyzer: An open platform for controllable mobile network. In *Proc. ACM MobiSys*, 2015.

[39] A. Patro, S. Rayanchu, M. Griepentrog, Y. Ma, and S. Banerjee. Capturing mobile experience in the wild: A tale of two apps. In *Proc. ACM CoNEXT*, 2013.

[40] C. Pei, Y. Zhao, G. Chen, R. Tang, Y. Meng, M. Ma, K. Ling, and D. Pei. WiFi can be the weakest link of round trip network latency in the wild. In *Proc. IEEE INFOCOM*, 2016.

[41] A. Razaghpanah, N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, P. Gill, M. Allman, and V. Paxso. Haystack: A multi-purpose mobile vantage point in user space. Technical report, arXiv:1510.01419, 2016.

[42] P. R. Rider. Variance of the median of small samples from several special populations. 55(289):148–150, 1960.

[43] S. Rosen, H. Luo, Q. A. Chen, Z. M. Mao, J. Hui, A. Drake, and K. Lau. Discovering fine-grained RRC state dynamics and performance impacts in cellular networks. In *Proc. ACM MobiCom*, 2014.

[44] S. Rosen, H. Luo, Q. A. Chen, Z. M. Mao, J. Hui, A. Drake, and K. Lau. Understanding RRC state dynamics through client measurements with Mobilyzer. In *Proc. the 6th Annual Workshop on Wireless of the Students, by the Students, for the Students (S3)*, 2014.

[45] J. P. Rula and F. E. Bustamante. Behind the curtain: Cellular DNS and content replica selection. In *Proc. ACM IMC*, 2014.

[46] S. Sarat, V. Pappas, and A. Terzis. On the use of anycast in dns. In *Proceedings of 15th International Conference on Computer Communications and Networks*, pages 71–78, Oct 2006.

[47] A. Shaikh, R. Tewari, and M. Agrawal. On the effectiveness of DNS-based server selection. In *Proc. IEEE INFOCOM*, 2001.

[48] J. Sommers and P. Barford. Cell vs. WiFi: On the performance of metro area mobile connections. In *Proc. ACM IMC*, 2012.

[49] S. M. Stigler. Studies in the history of probability and statistics. XXXII: Laplace, fisher and the discovery of the concept of sufficiency. 60(3):439–445, 1973.

[50] K. Sui, M. Zhou, D. Liu, M. Ma, D. Pei, Y. Zhao, Z. Li, and T. Moscibroda. Characterizing and improving WiFi latency in large-scale operational networks. In *Proc. ACM MobiSys*, 2016.

[51] Y. Sun, Y. Zhao, Y. Su, D. Liu, X. Nie, Y. Meng, S. Cheng, D. Pei, S. Zhang, X. Qu, and X. Guo. Hotspot: Anomaly localization for additive KPIs with multi-dimensional attributes. *IEEE Access*, 6:10909–10923, 2018.

[52] N. Vallina-Rodriguez, A. Auçinas, M. Almeida, Y. Grunenberger, K. Papagiannaki, and J. Crowcroft. RILAnalyzer: A comprehensive 3G monitor on your phone. In *Proc. ACM IMC*, 2013.

[53] N. Vallina-Rodriguez, N.Weaver, C. Kreibich, and V. Paxson. Netalyzr for Android: Challenges and opportunities. In *Proc. Workshop on Active Internet Measurements (AIMS)*, 2014.

[54] N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, N. Weaver, and V. Paxson. Beyond the radio: Illuminating the higher layers of mobile networks. In *Proc. ACM MobiSys*, 2015.

[55] Z. Wang, J. Huang, and S. Rose. Evolution and challenges of DNS-based CDNs. *Digital Communications and Networks*, 4(4):235 – 243, 2018.

[56] S. Wassermann, J. P. Rula, F. E. Bustamante, and P. Casas. Anycast on the move: A look at mobile anycast performance. In *Proc. TMA*, 2018.

[57] D. Wu, R. K. C. Chang, W. Li, E. K. T. Cheng, and D. Gao. MopEye: Opportunistic monitoring of per-app mobile network performance. In *Proc. USENIX Annual Technical Conference (ATC)*, pages 445–457, 2017.

[58] Q. Xu, J. Huang, Z. Wang, F. Qian, A. Gerber, and Z. M. Mao. Cellular data network infrastructure characterization and implication on mobile content placement. In *Proc. ACM SIGMETRICS*, 2011.

[59] 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 *Proc. IEEE INFOCOM*, 2012.