A Machine Learning Based Approach to Mobile Network Analysis

Zengwen Yuan†, Yuanjie Li†, Chunyi Peng‡, Songwu Lu†, Haotian Deng‡, Zhaowei Tan†, Taqi Raza†
† UCLA Computer Science, Los Angeles, CA 90095
‡Department of Computer Science, Purdue University, West Lafayette, IN 47907

***Abstract*—In this paper, we present our recent work in progress on 4G mobile network analysis. In order to provide an in-depth study on the closed network operations, we advocate a novel approach via two-level, device-centric machine learning that can open up the system behaviors and facilitate fine-grained analysis . We describe our proposed approach, and use the latency analysis on two popular mobile apps (Web browsing and Instant Messaging) to illustrate how our scheme works. We further present preliminary results and discuss open issues.**

I. Introduction

In recent years, mobile access to the Internet via the 4G LTE network has become the norm rather than exceptions for billions of global users. To fully understand and refine the system design and operations, fine-grained network analysis has been an important topic. Take the latency analysis as an example. On one hand, latency plays a critical role for many apps and greatly affects user experience. Users are typically sensitive to latency, and tend to abort the Web access or decrease the usage frequency on instant messaging (IM) if the perceived latency is high. On the other hand, latency is also important for operators and service providers. Amazon claimed that every 100 ms latency would cost them 1% in sales [1], while Google found an extra 500 ms in search page generation time killed user satisfaction and dropped 20% in traffic [1]. In a KissMetrics study [2], 73% of mobile internet users say that they have encountered a Website too slow to load, with 1-second delay in page response results in 7% reduction in conversions.

In this paper, we identify issues on mobile network analysis, as well as limitations with the convention approaches, which are either problem specific or infrastructure based. Instead, we make a case for a two-level, device-centric approach via machine learning. In our approach, each mobile device will sense the mobile network by collecting the runtime data traces and apply device-specific, local machine learning algorithms on network analysis. The device subsequently shares the analysis results to the cloud processor for aggregated processing. Overall, each device can obtain fine-grained, runtime view through its local analysis, where the cloud processor offers coarse-grained, global view across multiple devices and geographic regions.

We further illustrate our methodology in a case study on latency analysis for mobile apps over 4G LTE networks. We assess two popular apps, Web browsers (e.g., Safari on iphones and Chrome on Android phones) and instant messaging (IM) apps (e.g., WhatsApp), over four US LTE carriers. Our study can first provide fine-grained analysis at each device. It uncovers all sources of latency for mobile data access, including control-plane operations, data-plane signaling, and data-plane packet forwarding. For control-plane latency, the problem is that, LTE frequently tears down the established data access, to save energy at the device and release allocated radio resource at the network. Every time the application requires data access, LTE must rerun its control-plane procedures to establish this access, thus incurring considerable latency due to the signaling exchanges and connectivity state transfer. Whenever the app generates bursty traffic, this delay component is likely to become a main source for latency. For data-plane latency, signaling procedures to allocate time slots over the radio link contribute to non-negligible latency, whereas packet transmissions and retransmissions will also take time to complete. Once the user roams, handover will incur additional latency for the mobile apps.

Given the local results from each device, the high-level synthesizer can further reveal the global results across devices. We have three concrete findings: (1) LTE radio connectivity is short-lived and triggers frequent data access setup (every $133.6\,\mathrm{s}$ on average). It is rooted in the LTE control-plane design for radio resource control at the network side, and further aggravated by the phone-side power-saving mode; (2) The latency incurred by data access setup ranges from $147.9\,\mathrm{ms}$ to $196.3\,\mathrm{ms}$ on average for US mobile carriers. However, it may go up to $2.96\,\mathrm{s}$ in rare cases. Moreover, this overhead is insensitive to varying radio link quality; (3) Both radio connectivity setup and connectivity state transfer may be the dominating factors in LTE data access setup latency. Random access is the main bottleneck for radio connectivity setup, while multiple rounds of state exchanges among multiple network nodes contribute to the latter. These latencies contribute 42.3% of overall Web latency, and 51.4% of overall IM latency on average.

The rest of the paper is structured as follows. Section II introduces 4G LTE network background and identifies issues for its network analysis. Section III discusses limitations of the current infrastructure-based scheme, and makes a case for two-level, device-centric machine learning based approach. It further sketches out our proposed scheme. Section IV describes the detailed procedure on applying the approach on the case study of latency analysis. Section V presents the preliminary analysis results by using the procedure. Section

VI discusses the possible solutions and future issues. Finally, Section VII concludes the paper.

II. Issues on Mobile Network Analysis

A. Running Mobile Apps over 4G Networks

We use two popular mobile applications, Web browsing and instant messaging (IM), to illustrate how mobile apps work over the 4G LTE network.

Figure 1 exemplifies the step-by-step procedures for mobile Web access over LTE (IM follows similar procedures). In Step (a), the Web browser (e.g., Safari) sends its HTTP request to the mobile OS, which subsequently delivers the IP packets to the LTE interface in Step (b). Assuming that the phone initially stays in the idle mode (i.e., no data transfer), no IP packets can be delivered until data access is first established over LTE (mandated by the 3GPP standards). To this end, the LTE chipset on the phone starts to initiate new data access setup with the LTE network in Step (c). This typically requires several functions of radio connectivity setup between the device and the base station, security key derivations for the new data session, installing states at the device and the base station, and optional authentication between the device and the network. Therefore, signaling exchanges may be needed over multiple nodes (the base station, the mobility controller, and the user profile server) on the control plane. Upon completing Step (c), data access is established over LTE. In Step (d), the LTE interface on the phone can now start to deliver data packets, which traverse the LTE network (base stations, gateways) and the wired Internet to reach the Web server. Upon receiving the HTTP request, the server replies with the HTTP response message, which traverses the reverse path to reach the mobile application in Steps (e)(f).

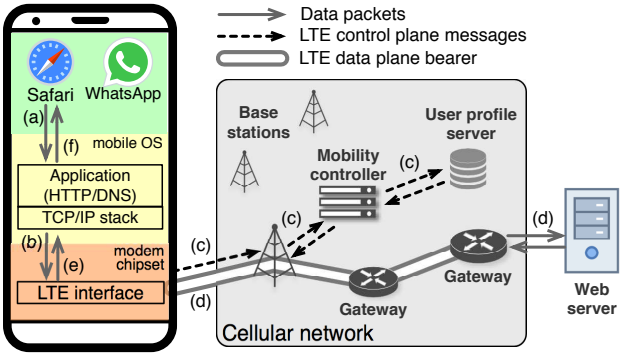


Fig. 1: Mobile apps access servers via LTE network.

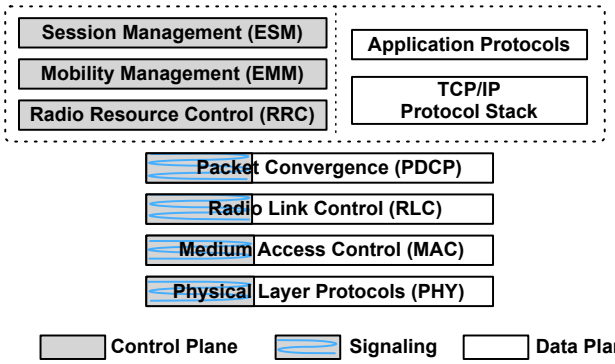


Fig. 2: LTE protocol stack for both control and data planes.

Note that the above data access involves both control-plane procedures (Step (c)) and data-plane operations (Step (d)). As illustrated in Figure 2, the control-plane operations follow a layered protocol stack, and they together provide vital control functions such as mobility support, radio resource control, security, etc. Also shown in Figure 2, the data plane operations include actual data forwarding, as well as the signaling to allocate radio channels. That is, the data-plane procedures can be further split into the packet-forwarding operations and the signaling operations (i.e., radio channel resource allocation of time slots). Moreover, the operations differ for static setting and mobility cases, i.e., handoffs are needed for mobility support. We will further illustrate these details later.

B. Issues on Fine-Grained Network Analysis

We next identify four issues that make network analysis over such mobile networks be difficult using the conventional approach.

1) Tightly-guarded system operations: Different from the Internet, current 4G network operations and runtime behaviors are not accessible to researchers and end users. While operators can collect the data traces from the infrastructure side, they carefully safeguard their access to the research community. As a result, 4G mobile network operations remain closed to users and researchers, thus hindering fast research progress.
2) Access barriers to both hardware and software stack: While the users can access certain mobile OS (e.g., Android), their access to the LTE hardware, as well as the software stack, is mostly prohibitive. The research community cannot access the radio hardware, or reconfigure the parameters, or test with new functions in general.
3) Complexity in data and control planes: Different from the Internet, 4G networks run much more complex functions on both planes. On the control plane, they provide more signaling features beyond the Internet counterpart, including mobility support, radio resource management, security, etc. On the data plane, in addition to packet forwarding, they also run signaling operations to allocate time slots for packets over the radio channel.
4) Distributed operations across multiple protocol layers. The mobile network operations also span distributed entities, including devices, network infrastructure, mobile OS, and apps. At each entity, they further operate across multiple layers beyond the Internet TCP/IP protocol stack. on both control and data planes

III. A Case for Device-Centric, Two-Level ML-Based Approach

A. State-of-the-Arts and Limitations

The fine-grained 4G network analytics today are largely *infrastructure-based*. These platforms are usually owned by the 4G LTE operators (such as AT&T and T-Mobile), and deployed in operators' data centers. They collect the mobile network data from the 4G LTE infrastructure (base stations, gateways, and controllers), aggregate them over short time intervals, and perform analytics over these aggregated data. The state-of-the-art analytics platforms can perform the control and data-plane analysis [3], [4]. But in reality, the data-plane analysis is usually enabled for a short period (*e.g.* for troubleshooting) due to the huge data packet volumes [3].

The infrastructure-based analytics have three fundamental limitations: (1) *Not scalable*: The infrastructure-based solutions are not scalable to massive data. In particular, the existing solutions cannot support the large-scale data-plane analysis due to the huge data volume; (2) *Incomplete dataset*: The infrastructure-based analytics cannot directly collect the data of user-perceived QoEs. Although indirect inference is possible (via DPIs), it can be inaccurate and even misleading; (3) *Opaqueness*: The operator-owned mobile network analytics is not accessible to researchers and mobile app developers. The operators are reluctant to share their infrastructure-side data or analytics with the community.

B. The Promise of Device-Centric ML Approach

This paper explpores a *device-centric* approach for mobile network analysis. Different from the infrastructure-based solutions today, our approach leverage the end devices to collect the network data, perform the local ML-based analysis, and crowdsource the analytics results to form a global mobile network knowledge base. This offers three unique benefits:

- *Scalability*: By offloading the most analysis to the end devices, our approach can support both the control and data-plane analysis, and scale out to huge data volumes and large geographical areas;
- *Direct access to QoEs*: The end device is the only place that can *directly* observe the user-perceived experiences-of-qualities (QoEs). Our approach can naturally support the correlation analysis between QoEs and the underlying 4G LTE network behaviors. This would yield new insights on how mobile network impacts the QoEs;
- *Availability*: Rather than relying on the network operators, the researchers and mobile app developers can readily perform the analytics using their own devices.

C. Our Two-Level Approach

Our device-centric analysis uses a two-level framework. At the *local level*, it senses the mobile network data inside each smartphone via hardware-software coordination. After the data pre-processing, it runs local ML algorithms to infer the control-plane protocol operations, and predict the data-plane performance and reliability. Next, our approach moves toward the *global level*: It crowdsources the analysis results from massive phones, and extracts feedback for smartphones to take intelligent adaptations. We next elaborate each level.

Local analysis: At this level, we run the mobile network data collection, pre-processing, and ML algorithms *inside* each device. It takes three steps:

1. Sensing the "black-box" mobile network: For comprehensive analysis, we collect the runtime, full-stack (from physical to app layer) mobile network data from two sources:

- **Above-IP network data**, which includes all the packets from the TCP/IP protocol stack. Such information can be readily obtained through the classical tcpdump;
- **Below-IP network data**, which includes the over-the-air control/data-plane signaling messages in 4G LTE. These messages carry rich network information, such as the wireless communication, link-layer data transfer, mobility management, and session management. Different from the above-IP data, these messages are not readily accessible inside the user space. To solve it, we have built an in-phone tool (MobileInsight [5]) via software-hardware collaboration: By leveraging a side channel (/dev/diag) across the software-hardware boundary, MobileInsight exposes over-the-air 4G LTE messages to the user space at runtime.

2. Automated data pre-processing: For accurate and efficient analysis, we pre-process the above raw data in two steps:

- **Data cleaning:** We eliminate the noisy items from the raw network data. For the above-IP data, we remove the TCP/IP packets that do not belong to the target app (*e.g.* background push notification data). For the below-IP data, we filter the over-the-air signaling messages that are of interest.
- **Data integration:** Given the data from two sources (tcpdump and MobileInsight), we need to align them before running the analysis. Specifically, for each TCP/IP packet, we should map it to the underlying 4G LTE signaling messages that are responsible for its delivery. This is realized by aligning these data based on timestamp and packet/message size. §IV will elaborate how this works.

3. Local ML-based analysis: We next run domain-specific ML algorithms to analyze the network performance and reliability. These ML algorithms are classified into two categories:

- **Control-plane ML:** At the control plane, we aim to predict the signaling protocol operations that would affect the data performance. For instance, the control-plane session setup and migration (in device mobility) would incur extra data access latencies (§V). To predict them, we observe that many signaling protocol operations are regulated by the standards [6], [7], and are highly *interactive* and *stateful*. We thus adopt the graphical ML models for the control plane operations. Each signaling protocol is modeled as a finite state machine. We first infer the state machine structure based on the standards and graphical ML algorithms (such as Bayesian network and state machine merging). Then to predict the protocol operations, we traverse the inferred state machine based on the runtime

signaling messages, and predict the occurrence of the next state transition. As an example, we have developed a handoff prediction algorithm based on this graphical model [5].

• **Data-plane ML:** At the data plane, our goal is to predict the data transfer performance, and infer the failure causes (if any). Different from the control plane, the data-plane performance and failures may not be appropriately modeled as finite state machines. Instead of the graphical models, we have developed regression-based models in various scenarios. For example, iCellular [8] uses the regression tree algorithm to predict the runtime performance of different mobile network operators. In [9], we have built a predictor of the link-layer ACK/NACK flip based on the timer estimations.

Global analysis: Beyond the local analysis, we move one step further to scale out the mobile network analytics. This is realized through the cloud-based post-processing system [10]. It first collects the local analysis results from massive mobile devices. With the temporal/spatial aggregation, we can obtain a global knowledge base of the mobile network that covers large geographical areas, various mobile operators and phone models. This global knowledge base can offer crowdsourced feedback for mobile devices, such as ranking the best mobile network operators, and discovering the better phone settings for the network performance and reliability.

IV. Case Study on Latency Analysis: Procedures

We now present an in-depth study on latency analysis on mobile apps over 4G LTE networks. We first present our methodology that follows our approach above.

Usage setting We run two popular mobile apps. For Web apps, we run two most popular Web browsers at the mobile device: Safari on iOS and Chrome on Android. We visit a small web page when the phone is in the idle mode. The Web page is 4KB in size with only plain HTML tags. For each mobile carrier, the experiment is repeated 50 times under different signal strengths and locations. The tcpdump trace is used for cross validation.

We test WhatsApp (version 2.17.146) latency on Nexus 6P (Android 7.1.1). We send an instant message every 20 seconds. The recipient's phone screen is turned off when receiving the message. For each mobile carrier, we repeat the experiment for at least 50 runs to obtain the average result unless otherwise specified.

Data sensing We record event timings at three levels. We first log the application-level timings to quantify the user-perceived latency. For Web browser, we adopt the Navigation Timing API [11] to record the precise Web browsing timestamps. It provides accurate timing for web browsing events, including the OS overhead, the DNS time, the TCP connection setup time and page rendering time. For IM app, we use its internal message database, which logs the epoch time of each message being sent[1]. We then use TCP/IP level logging (tcpdump) and LTE chipset logging (MobileInsight) to record the mobile OS and LTE signaling events. These logs enable the detailed breakdown analysis on data access setup latency. Figure 3 illustrates the timing and logging model for data access setup latency. During the experiment, we constrain the background traffic from other apps to quantify the latency impact by the assessed applications.

[1]For WhatsApp, we use its `msgstore.db`.

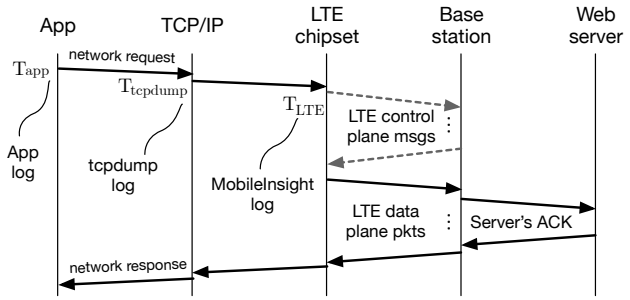


Fig. 3: Logging timestamp alignment.

Local processing at each device We further take the generic approach at each device for efficient local processing as follows.

a) Timestamp alignment: To correlate the app latency with the underlying LTE operations, we need to align the timings for all three levels of logs. We first match the app-level epoch to the tcpdump record using specific protocols. For Web browsing, we match the app's timing of the DNS lookup event (or TCP connection SYN packet) with its counterpart in tcpdump timing to obtain the timing offset. The app timing and the TCP timing may have a gap. We find that such gap is usually negligible in reality (usually under 1 ms). The reason is that, the applications we test are written as native apps, with marginal overheads from app to network layer.

To align the timing events in the MobileInsight logs with tcpdump logs, we observe that IP packets (from tcpdump) are encapsulated in the data frames[2] over LTE. LTE adds a fixed-sized frame header (2 bytes) to each IP packet. We use this linear relation between the PDCP packet recorded by MobileInsight and the IP packet logged by tcpdump to obtain the timing offset.

b) Latency breakdown: We next perform a breakdown analysis on the overall latency. We divide the standardized procedures, quantify each component, and identify the main bottleneck.

Figure 4 summarizes the control-plane procedures when establishing LTE data access (standardized in [13], [6], [14]). The device first sets up its radio connectivity to the base station. It then sends the data access request to the mobility controller using this dedicated radio connectivity. Optionally, the network and the phone perform mutual authentication. The base station then fetches the per-device connectivity states from the controller, which carries critical information for data access (device ID, QoS profile, and derived security keys, etc.).

[2]The frames are encapsulated by LTE Packet Data Convergence Protocol (PDCP) [12].

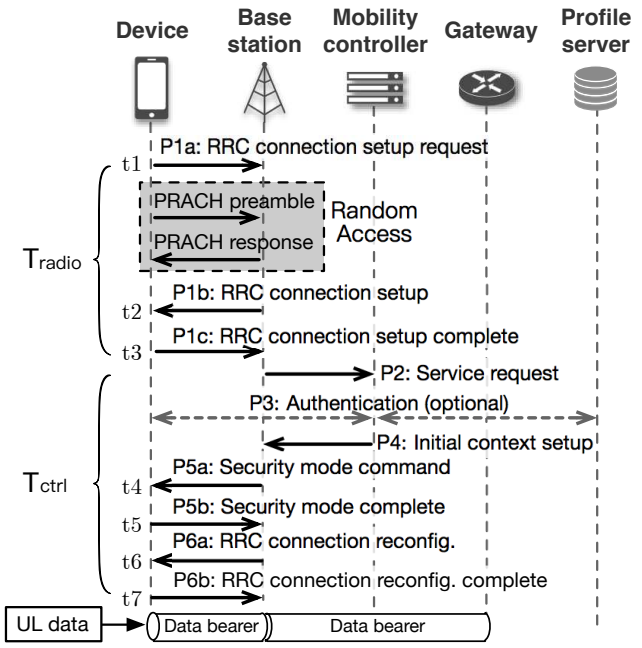


Fig. 4: Control-plane procedures for LTE data access

Last the base station enables the security mode to encrypt the data over wireless, reconfigures the connectivity based on these states, and start the data delivery.

The issue for the breakdown analysis is that, certain procedures inside the LTE network are not directly visible to the end device. To address it, we correlate the core-network operations with the device-side traces by following LTE standards [13], [6], [14]. Figure 4 shows how it works. For each data access request, we record the timestamps for the events of radio connectivity setup (t_1, t_2, t_3), radio-level security mode command (t_4, t_5) and radio bearer setup (t_6, t_7). The total data suspension time is thus $T_{total} = t_7 - t_1$, and the radio connectivity setup latency is $T_{radio} = t_3 - t_1$. The control-plane latency for establishing data access is $T_{ctrl} = t_7 - t_3$. It includes the state transfer from the mobility controller to the base station, the security mode activation, and the connectivity reconfiguration.

c) ML scheme: We further apply ML algorithms to enable latency prediction for application adapting to dynamic network conditions, and root cause analysis for the unexpected, large latency factors.

Latency prediction at runtime helps application to adapt to dynamic network conditions, and potentially improve performance. On control plane latency, we target the procedures that may incur significant latency, e.g. handover. When handover occurs, web access requests are delayed and sometimes propagate to application layer causing retransmission or timeout. Therefore, predicting the handover latency could help application reschedule its data request, if a handover is imminent.

We predict handover using a decision tree classifier. We are able to engineer domain specific features (such as LTE control message that are exchanged immediately before handover occurrence, including "measurement report", "mobility control info" and "measurement config". The features are binary encoded, 1 for message exists and 0 for the opposite case. We also label the dataset indicating a handover is about to happen if cell ID changes within a window size. We do this because we know that these specific messages are related to a handover regulated by 3GPP standards. We then train the decision tree to detect these messages. The trained decision tree has a maximum leaf count of 15 and class weight of 0.2 for non-handover and 0.8 for handovers. The predict result achieving 99.9% accuracy. Last, we fuse the local latency result on handover with the prediction output of the decision tree, yielding an estimate of upcoming handover latency.

On data plane, we focus on the frequently incurred NACK/ACK flip at MAC layer [9]. Part of the LTE link-layer error recovery mechanism, speeding up the detection of NACK/ACK flip could reduce the downlink retransmission latency. We use timer-based regression for early NACK-ACK flip detection. The approach relies on the domain-specific insight that the base station takes shorter time to process NACK than ACK by LTE design. We maintain and update a device-side timer for the round-trip of NACK, based on previous NACK-triggered retransmissions using exponential moving average. When a NACK is sent, the timer starts. If no response before the timer expires, the algorithm recognizes that the base station may receive a flipped NACK as ACK.

Root cause analysis of large latency factors provides starting points on how the app performance may be improved. It is enabled and bootstrapped by human labeling of a set of diagnosis cases. We leverage domain knowledge to label causes of long latency in data plane, for example, packet loss at RLC layer, ACK/NAK flip at MAC layer, attach failure, radio link failure etc. With each instance of long data plane latency, we map it to one or a few of the causes mentioned. We then construct a Bayesian network to capture the causal relation between the instance (under what conditions such case occur) and the diagnosis. Finally, with the parameters learned, we can infer the maximum a posteriori probability (MAP) of different causes on a long latency case.

Synthesizer Our next step is to perform aggregate processing by using the feedbacks from multiple individual devices. We collect local latency analysis results of common latency components, such as service request latency at control plane, grant waiting time at data plane, or long latency caused by various failure cases. Then we aggregate all these data points and perform aggregated analysis in the cloud and further generate new insights. For example, we aggregated LTE service request latency by radio signal strengths, and find that it is insensitive to varying radio link quality (Figure 8). This helps our latency prediction in ML scheme as the radio link quality is not an effective feature in prediction. We can further group latency performance by geographical area, by operators, by device model, by OS build etc. Based on these result, we have built a cloud knowledge platform that enables runtime, fine-grained cellular analytical result query

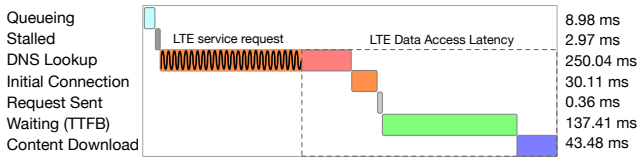


Fig. 5: Latency breakdown for an example webpage access.

for latency based on geographical area [10]. This helps in-depth analytics in understanding the performance difference observed under different conditions.

V. Learning Latency: An Example

We provide an example in this section on how to obtain an in-depth understanding of web access latency in LTE, following a top-down approach and with the help of global view. We first breakdown, for both the individual latency of both data and control plane and aggregated latency components.

A. App Perceived Latency

As the showcase, we visited a lightweight Web page (https://www.bing.com) using latest version of Chrome on Android, and recorded the timing breakdown by Chrome DevTools. In one example run, we recorded total $473\,\mathrm{ms}$ latency for loading the index page of the website (26.4 KB) under good LTE signal strength (-95 dBm). This is the total elapsed time from the URL request sent to the whole Web page is downloaded and rendered. The Chrome DevTool performed a coarse breakdown of major events shown in Figure 5, following the HTML resource timing model detailed in [15]. However, this breakdown does not give sufficient insights as to why the two major components are taking long time: DNS ($250\,\mathrm{ms}$) and waiting time ($137\,\mathrm{ms}$). To make better understanding, we performed latency analysis at device-side first.

B. Two-Step Breakdown at Device Side

We take a two-step approach on device side, by assessing control plane latency first and analyze data plane latency next. This is due to the fact that LTE control plane operations, such as Service Request, precede data forwarding in each data session. We can further analyze any failure or loss case using machine learning based approach. Such analysis work flow for this example applies to other web access requests as well, as both control plane and data plane procedures here are repetitively involved. This method is purely automated and maps both control plane and data plane latency in LTE on the fly.

Control plane latency We single out LTE control plane latencies from data access latency. By automatically matching the LTE control message sequence, we were able to identify LTE Service Request procedure from the trace. As Figure 5 shows, we separated the LTE service request procedures (squiggled within the DNS request bar) from the DNS request, which incurs $172\,\mathrm{ms}$ latency. This was invisible to the app-level latency breakdown without below-IP layer breakdown.

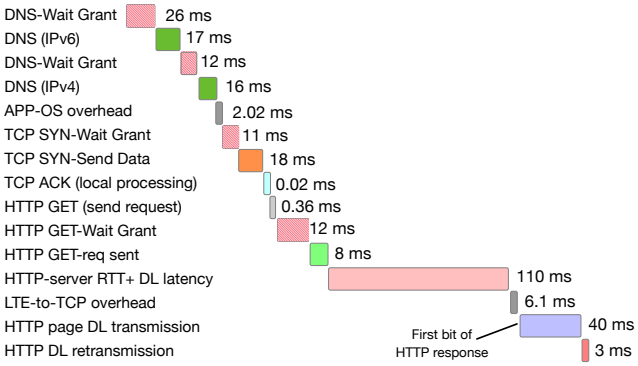


Fig. 6: Data access latency breakdown.

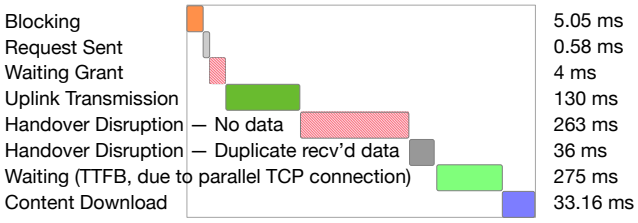


Fig. 7: Latency breakdown for a handover event.

We may even further breakdown the radio RTT time and processing time at the network side for this control plane latency.

Data plane latency We next zoom in and focus on the data access latency (dotted box in Figure 5, including DNS, TCP and HTTP requests). As the LTE enforces fine-grained radio resource scheduling, each uplink data request has some waiting time for allocated radio resource grant. We thus was able to match such latency using the data-plane message exchange between the phone and the base station at PHY layer. We count the time elapsed using the scheduling request message and the grant allocation received, plus the mandatory $4\,\mathrm{ms}$ waiting time before the data can be pushed to the air interface [16]. In the object request, we mapped three waiting latencies for LTE grant: before each DNS request ($26\,\mathrm{ms}$ and $12\,\mathrm{ms}$), between DNS response and TCP SYN packet ($11\,\mathrm{ms}$), and between TCP SYNACK and first HTTP GET request ($12\,\mathrm{ms}$). Figure 6 illustrates the details.

Latency mapping for failures In some cases, failure may occur at either control plane or data plane. For instance, when handover occurs, the data plane is suspended not only due to radio reconnection period, but also caused by head-of-line blocking due to duplicate data. In such case, automated pattern matching also worked well in breaking down the total data access waiting time. LTE control messages help count the disruption of handover waiting time, while duplicate data detection at data plane identifies head-of-line blocking contribution. In the example illustrated in Figure 7, the total $708\,\mathrm{ms}$ waiting time during user mobility was breakdown into four components: Uplink transmission, handover disruption (radio reconnection, no data), handover (duplicate data), and

true waiting time.

C. Results on Crowdsourced Analysis

We next describe the crowdsourced latency analysis results at the synthesizer.

Crowdsourced setting Our case study is based on 15 volunteers (college students, faculty members and company employees) using 23 phones. They cover seven Android phone models (Google Pixel, Huawei Nexus 6P, Motorola Nexus 6, Samsung Galaxy S4/S5, LG Optimus 2, and LG Tribute) and two iPhone models (iPhone 5 and iPhone 6s Plus). They further include all four major US carriers (Verizon, AT&T, T-Mobile, Sprint and a virtual operator, Google Project Fi [17]). Overall, the study uses 95,057 data sessions that require data access setup via 4G LTE signaling.

Latency by data access setup We next show that, each LTE data access setup incurs a variable amount of delay. Moreover, this latency varies across different mobile carriers. Figure 8 plots the results for five US mobile operators. We quantify the latency as the elapsed time of between each data access request and the instant when the device can send/receive data. The analysis reveals that, given each data access request, the suspended data access ranges from $77.4\,\mathrm{ms}$ to $2956.0\,\mathrm{ms}$. Among all samples, the average overall latencies due to this control-plane operation are $196.3\,\mathrm{ms}$, $165.5\,\mathrm{ms}$, $147.9\,\mathrm{ms}$, $153.6\,\mathrm{ms}$, and $163.2\,\mathrm{ms}$ in AT&T, T-Mobile, Sprint, Verizon, and Project Fi, respectively.

Another interesting observation is that, the latency incurred by the data access establishment is *insensitive to varying radio link quality*. Figure 8 shows that, the correlation between the signal strength and the average control-plane latency is weak. Instead, as a control-plane procedure, the latency due to data access request is primarily determined by the signaling operations. We can further validate this with a breakdown analysis.

Latency breakdown Figure 9 plots the latency breakdown results for all four operators. It shows that, the overall latency is dominated by two components:

(1) T_{radio}*: Radio connectivity setup (w/ random access).* Among all the samples, the radio connectivity setup contributes $67.5-1665.0\,\mathrm{ms}$ of the overall LTE access latency. On average, it contributes 39.7%, 44.0%, 61.9%, 64.2% and 43.7% of total latency in T-Mobile, AT&T, Verizon, Sprint and Project-Fi, respectively. It turns out that, such long latency is mainly induced by the random access procedure. To send the first signaling message (radio connectivity setup) to the base station, the phone has to contend for the shared signaling channel[3]. This procedure can contribute up to 79.2% ($65\,\mathrm{ms}$ out of $82\,\mathrm{ms}$) of the radio connectivity setup time.

(2) T_{ctrl}*: Connectivity state transfer.* It contributes $28.75\,\mathrm{ms}$ to $2286.25\,\mathrm{ms}$ of the overall latency among all samples. On average, it contributes 60.3%, 56.0%, 38.1%, 35.8% and 56.3% of total latency in T-Mobile, AT&T, Verizon, Sprint and Project-Fi, respectively. It takes long time since it involves multiple network nodes and roundtrips. The base station has to fetch the connectivity state from the mobility controller, derive the security keys out of the state, enable the security mode to the device, and reconfigures the radio connectivity.

[3]LTE Physical Random Access Channel (PRACH) [18].

Impact on apps We find out that, the LTE data access setup is a dominating factor by contributing 42.3% of the total Web loading time on average, i.e., about $174.4\,\mathrm{ms}$. Therefore, TCP connection setup and DNS lookup will be affected by this data access request, since they are the initial two steps in Web loading. TCP connection also setup takes the largest portion, with LTE control plane latency contributing 62.4% ($176\,\mathrm{ms}$) of the TCP connection time ($282\,\mathrm{ms}$) in Safari. Chrome has similar results. For DNS lookup, its latency is affected by LTE's data access request only for the first time. Later DNS lookups will be met by phone's local cache, thus unaffected by LTE.

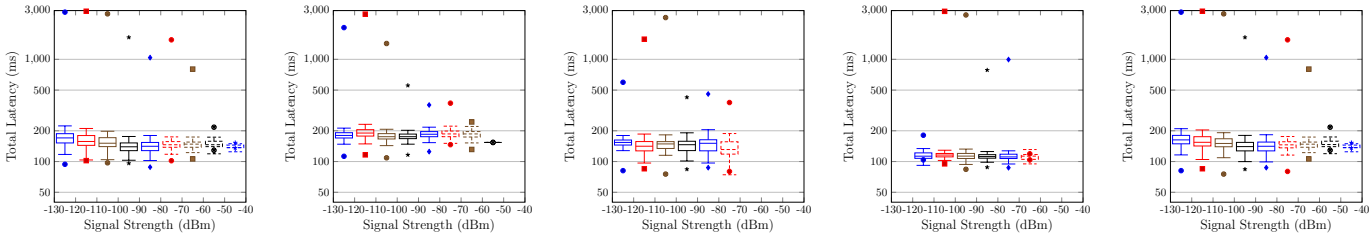
For IM, the LTE control-plane procedure is also the main factor by contributing 51.4% of the latency perceived by each IM message. The total time of sending out the first data packet and receiving server's ACK is $341.3\,\mathrm{ms}$. During that time, the LTE data access setup is trigged and incurs $175.4\,\mathrm{ms}$ extra latency on average. However, different from the Web app, this procedure does not affect the TCP connection setup. Instead, it affects data delivery in IM apps. IM apps (*e.g.* WhatsApp, Messenger, WeChat) may use encryption protocols (*e.g.* SSL or TLS) over TCP to deliver message. The TCP setup has been done once the app launches, thus being affected by LTE only the first time. They keep an always-on TCP connection. Upon sending new messages, LTE data access setup takes more than 50% total time to deliver the packet on average.

VI. Discussions

The above results also shed light on how to reduce latency. We describe both short-term fixes and long-term solutions. In addition to refining data-plane designs, it is equally important to renovate the control-plane operations for low-latency network access. The key insight is that, the device-centric ML analysis can provide useful feedback to reduce both control-plane and data-plane latencies. We discuss how it offers useful insights, and some unresolved issues that will be explored in the future.

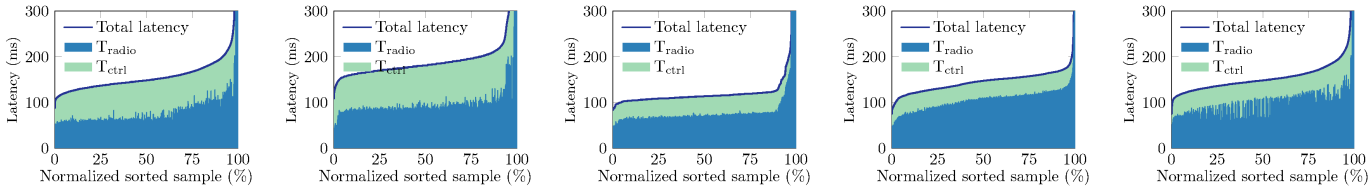
Data-plane work-arounds. Without signaling protocol updates or infrastructure changes, the client can still mask the LTE's data access setup latency. The idea is to keep the device in the *connected* state over time, thus preventing repetitive LTE data access setup. For example, the apps may regularly send keep-alive messages to avoid entering the idle state. The cost is on the extra power waste due to the always-on connectivity. This energy cost can be mitigated if accurate traffic prediction is feasible. Our recent study [9] has made a case to demonstrate its feasibility, by developing the domain-specific data-plane ML algorithms.

Control-plane protocol acceleration. In the long run, the control-plane procedures should be optimized for low latency. The device-centric control-plane ML can also provide useful



(a) T-Mobile (b) AT&T (c) Sprint (d) Verizon (e) Project Fi

Fig. 8: Overall latency for establishing 4G LTE data access. Dots represent max/min value for each group.



(a) T-mobile (b) AT&T (c) Verizon (d) Sprint (e) Project Fi

Fig. 9: Breakdown of 4G LTE latency in signaling operations for data access setup.

optimization hints. In particular, connectivity state transfer between the base station and the mobility controller should be accelerated. For example, for static users served by the same base station, the base station can speed up the state transfer via caching or pre-fetching. The control-plane ML can help predict the status of the mobile devices and decide the appropriate caching/pre-fetching strategies.

Remaining issues and possible directions Our device-centric mobile network analysis is still in progress. We have identified some research challenges for further exploration. First, besides latency analysis, we plan to extend our device-centric ML approach to other network metrics (throughput, packet loss, reliability) and application-specific QoEs. Second, we intend to derive the theoretical bounds on our two-level ML based scheme. Specifically, we are interested in correlating the local accuracy at each device and the synthesizer's global error bounds. Third, we plan to explore the privacy issues in a more systematic way to ensure the analysis will preserve each user's privacy while offering accurate analysis results. Last but not the least, we are also in the process of exploring other AI techniques, such as deep learning and Bayesian Network Inference, on the root cause analysis of network behaviors.

VII. Conclusion and Future Work

The 4G LTE network is by far the only large-scale wireless infrastructure, in par with the wired Internet, which offers ubiquitous radio coverage and network access. Most users run their mobile apps over LTE on a daily basis. Therefore, mobile network analysis is important to help better understand and refine its design and operations.

In this work, we present preliminary results on our machine learning based approach to automated network analysis. We have used the case study on latency analysis to illustrate how our methodology helps to reveal various latency components that are unreported in prior work. Moreover, we believe that our two-level ML based scheme holds promise to unveil the tightly-guided operation issues over 4G/5G mobile networks that have been eluding from the research community.

Acknowledgment

We acknowledge the generous support of National Science Foundation (NSF) on the early stage of this work through awards (CNS-1526985, CNS-1423576, CNS-1748630, and CNS-1749049). Any opinions, findings, and conclusion or recommendations expressed in this work are those of the authors and do not necessarily reflect the view of NSF or the US government.

References

[1] GigaSpaces, "Amazon found every 100ms of latency cost them 1% in sales." https://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/.

[2] K. Blog, "How Loading Time Affects Your Bottom Line," https://blog.kissmetrics.com/loading-time/.

[3] A. Iyer, L. E. Li, and I. Stoica, "Celliq: real-time cellular network analytics at scale," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 309–322.

[4] A. P. Iyer, I. Stoica, M. Chowdhury, and L. E. Li, "Fast and accurate performance analysis of lte radio access networks," *arXiv preprint arXiv:1605.04652*, 2016.

[5] Y. Li, C. Peng, Z. Yuan, J. Li, H. Deng, and T. Wang, "Mobileinsight: Extracting and analyzing cellular network information on smartphones," in *The 22nd ACM Annual International Conference on Mobile Computing and Networking (MobiCom'16)*, New York, USA, Oct. 2016.

[6] 3GPP, "TS36.331: Radio Resource Control (RRC)," Mar. 2015. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/36331.htm

[7] ——, "TS24.301: Non-Access-Stratum (NAS) for EPS; ," Jun. 2013. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/24301.htm

[8] Y. Li, H. Deng, C. Peng, G.-H. Tu, J. Li, Z. Yuan, and S. Lu, "iCellular: Define Your Own Cellular Network Access on Commodity Smartphones," in *Accepted by USENIX NSDI*, March 2016, draft available: http://arxiv.org/abs/1510.08027.

[9] Z. Tan, Y. Li, Q. Li, Z. Zhang, Z. Li, and S. Lu, "Enabling mobile vr in lte networks: How close are we?" in *The 44th ACM Annual Conference of Special Interest Group on Measurement and Evaluation (SIGMETRICS'18)*, Irvine, California, USA, Jun. 2018.

[10] W. Huang, C. Zhou, Y. Li, S. Lu, X. Wang, and L. Fu, "cnicloud: Querying the cellular network information at scale," in *11th Workshop on Wireless Network Testbeds, Experimental evaluation & CHaracterization (WiNTECH 2017)*, Snowbird, Utah, US, Oct. 2017.

[11] W. W. Draft, "Navigation Timing Level 2," https://www.w3.org/TR/navigation-timing-2/.

[12] 3GPP, "TS36.321: Evolved Universal Terrestrial Radio Access (E-UTRA); Packet Data Convergence Protocol (PDCP) specification," Jun. 2014. [Online]. Available: http://www.3gpp.org/DynaReport/36323.htm

[13] ——, "TS23.401: General Packet Radio Service enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access," Dec. 2015. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/23401.htm

[14] ——, "TS36.413: S1 Application Protocol (S1AP)," Jun. 2015. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/36413.htm

[15] J. Garbee, "Understanding resource timing," https://developers.google.com/web/tools/chrome-devtools/network-performance/understanding-resource-timing.

[16] "3GPP Standard Specification," http://www.3gpp.org.

[17] Google, "Project Fi," https://fi.google.com/.

[18] 3GPP, "TS36.213: Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures."