MopEye: Monitoring Per-app Network Performance with Zero Measurement Traffic

Daoyuan Wu[§*], Weichao Li[†], Rocky K. C. Chang[†], and Debin Gao[§]
[§]School of Information Systems, Singapore Management University
[†]Department of Computing, The Hong Kong Polytechnic University
[§]{dywu.2015, dbgao}@smu.edu.sg, [†]{csweicli, csrchang}@comp.polyu.edu.hk

ABSTRACT

Mobile network performance measurement is important for understanding mobile user experience, problem diagnosis, and service comparison. A number of crowdsourcing measurement apps (e.g., MobiPerf [4, 6] and Netalyzr [5, 7]) have been embarked for the last few years. Unlike existing apps that use active measurement methods, we employ a novel passive-active approach to continuously monitor per-app network performance on unrooted smartphones without injecting additional network traffic. By leveraging the `VpnService` API on Android, MopEye, our measurement app, intercepts all network traffic and then relays them to their destinations using socket APIs. Therefore, not only MopEye can measure the round-trip time accurately, it can do so without injecting additional traffic. As a result, the bandwidth cost (and monetary cost of data usage) for conducting such a measurement is eliminated, and the measurement can be conducted free of user intervention. Our evaluation shows that MopEye's RTT measurement is very close to result of `tcpdump` and is more accurate than MobiPerf. We have used MopEye to conduct a one-week measurement revealing multiple interesting findings on different apps' performance.

CCS Concepts

•**Networks** → **Network measurement; Mobile networks;**

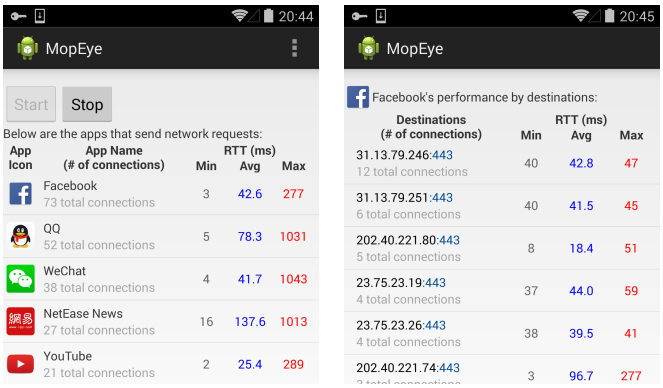
Keywords

Measurement Tool; Mobile Network Performance

*Most work by this author was performed at HK PolyU.

CoNEXT Student Workshop'15, December 1, 2015, Heidelberg, Germany.

DOI: http://dx.doi.org/10.1145/2842665.2843560



(a) An all-app view. (b) An individual-app view.

Figure 1: MopEye's user interfaces.

1. INTRODUCTION

In this paper, we propose a novel passive-active method to continuously monitor per-app network performance on unrooted smartphones without injecting additional network traffic. Specifically, we utilize the `VpnService` API available on both Android [9] and iOS [2] platforms for the measurement and implement it in MopEye (MObile Performance Eye), our Android measurement app. With the `VpnService` service, MopEye passively captures the traffic initiated by all apps and the return traffic. MopEye then forwards the captured IP packets to the remote servers using socket calls. Since MopEye directly communicates with the remote servers, it can estimate the round-trip time (RTT) based on the socket calls. As a result, no additional network traffic is ever incurred. MopEye also does not need the root privilege which is required for the `tcpdump`-based passive measurement. Figure 1 shows the two most important user interfaces of MopEye.

2. MopEye OVERVIEW

Figure 2 presents an overview of MopEye. Using the Facebook app as an example, we walk through the main steps for MopEye to measure its network performance.

1. **Packet capturing.** We leverage Android's `Vp-`

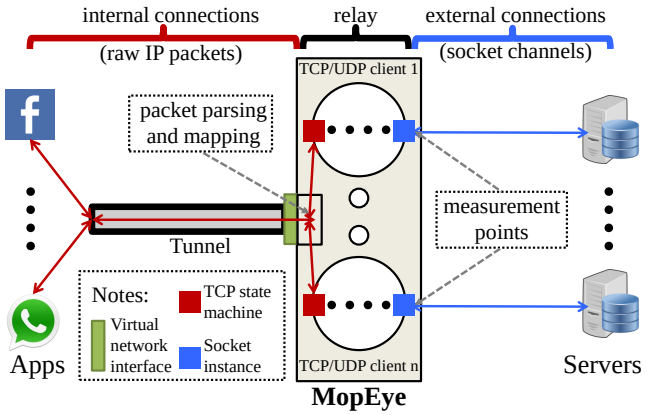


Figure 2: An overview of MopEye.

`nService` APIs to build a virtual network interface (green box in Figure 2). The interface enables MopEye to intercept all incoming and outgoing traffic of Facebook.

2. **Packet parsing and mapping.** MopEye then parses the captured packets to obtain the source and destination IP and TCP/UDP headers. Moreover, MopEye identifies the corresponding app for per-app measurement.
3. **Packet relaying.** MopEye works like a proxy in relaying traffic. As Figure 2 shows, MopEye maintains two separate connections with the Facebook server and the app, respectively. For the external connection with the server, MopEye allocates a TCP client object in memory and uses its socket instance to communicate with the server. For the internal connection with the app, as no TCP headers can be retrieved from the socket APIs, MopEye creates its own user-space TCP stack to manage a state machine, and then assembles and forwards packets to the tunnel. MopEye splices the two by cross-referencing each TCP client object and its state machine.
4. **Measurement methodology.** MopEye measures the RTT between itself and the Facebook server based on the sockets calls made for the external connection. Specifically, MopEye computes the `SYN-ACK` RTT from the `connect()` call. Experimental results show that only this call can accurately and stably reflect a single round of packet exchange.
5. **Measurement results.** As shown in Figure 1, MopEye displays the RTT results in all-app and individual-app views. It shows the number of connections made for each app since the beginning of the monitoring and reports the minimum, maximum, and mean RTTs.

We have overcome a number of challenges in the design and implementation of MopEye, which cannot be elaborated here due to the space limitation.

Table 1: The measurement accuracy comparison.

Destinations	MopEye (mean, in ms)			MobiPerf (mean, in ms)		
	`tcp dump`	Mop Eye*	δ	`tcp dump`	Mobi Perf	δ
Google (`216.58.221.132`)	4.26	4	**0**	4.29	16.4	**12.11**
	4.47	5.5	**1.03**	4.35	18.5	**14.15**
	5.32	5	**0**	4.85	18	**13.15**
Facebook (`31.13.79.251`)	36.55	37	**0.45**	36.39	59.5	**23.11**
	36.55	37	**0.45**	36.72	55.2	**18.48**
	38.54	38.5	**0**	46.10	63.2	**17.10**
Dropbox (`108.160.166.126`)	284.85	284.5	**0**	361.76	409.7	**47.94**
	390.94	391	**0.06**	388.94	411.5	**22.56**
	513.78	513.5	**0**	395.87	475.2	**79.33**

* We round MopEye's μs-level results to ms-level, e.g., 4.135ms to 4ms.

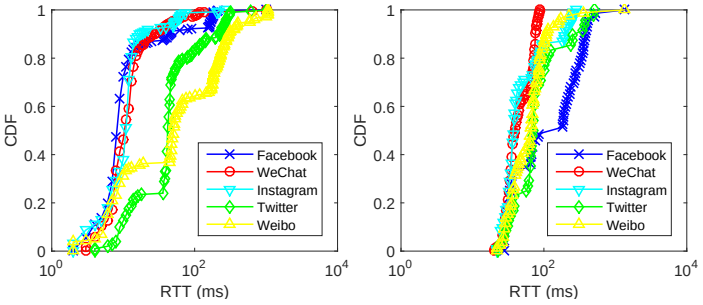
3. EVALUATION

3.1 Measurement Accuracy and Overhead

Measurement accuracy. We compare MopEye with MobiPerf v3.4.0 (the latest version at the time of our evaluation), which is powered by state-of-the-art Mobilyzer library [6]. We use MobiPerf's HTTP ping [3] for comparison because it also uses `SYN-ACK` for RTT measurement. We run `tcpdump` to provide the reference measurement results. In Table 1, we present three sets of results for Google, Facebook, and Dropbox, which have different ranges of RTTs. Each result is the mean of ten independent runs because MobiPerf does not provide detailed results of each run. The difference between results of `tcpdump` and MopEye/MobiPerf is denoted by δ. This table clearly shows that MopEye has a much better accuracy than MobiPerf — MopEye's measurement deviates from `tcpdump`'s by at most 1ms whereas MobiPerf's deviation ranges from 12ms to 79ms.

By comparing MobiPerf's codes with our MopEye's implementation, we identify three main contributing factors for MopEye's higher accuracy. First, MopEye uses the low-level socket `connect()` call, instead of the HTTP-level `HttpURLConnection.connect()` in MobiPerf. Second, the timestamps for MopEye are collected just before and just after the `connect()` call. In contrast, MobiPerf's measurement includes the overhead of invoking pre-connect functions, such as `openConnection()`. Third, MopEye employs the more accurate nanosecond-level timestamp method, rather than the millisecond-level `currentTimeMillis()` in MobiPerf.

Network delay overhead. It is important for MopEye *not* being a bottleneck to other apps. We thus measure the additional delay experienced by other apps when MopEye is active. We measure the overhead of the `SYN-ACK` packets using our measurement tool that invokes `connect()` to measure the connection time. By subtracting this delay by the MopEye measurement to the same destination, we obtain the delay overhead. For the data packets, we use the popular Ookla Speedtest app [8] to measure the delay with and without MopEye. Their difference is the overhead introduced by MopEye. Both experiments are repeatedly run on a Nexus 4 running Android 5.0. With 95% confidence interval, the



(a) Under WiFi. (b) Under 4G.

Figure 3: Top five apps' performance in our dataset.

mean delay overhead of a round of `SYN-ACK` packets is 4.15∼5.98ms, and that of data packets is 1.22∼2.18ms. Considering that the average RTT of the AT&T LTE network is about 75.47ms [1], we find the additional delay introduced by MopEye quite acceptable.

3.2 One-week Measurement Results

We performed a one-week measurement on a Nexus 5 in Hong Kong on May 2015. MopEye relays a total of 5,598 connections out of which 5,410 are successful connections (188 are nonresponsive ones due to, e.g., unavailable servers). Among the 5,410 RTTs collected, 4,025 are over WiFi with the remaining 1,385 over LTE 4G network.

We plot the RTT distribution for the five most popular apps in Figure 3(a) (for WiFi) and Figure 3(b) (for 4G). Facebook, WeChat, and Instagram achieve better performance than Twitter and Weibo in the WiFi network, as most of the RTTs (>85%) for these three apps are less than 20ms. Over 4G, however, Twitter and Weibo outperform Facebook which suffers from a significant performance degradation with the mean RTT risen from 26.1ms to 188.4ms. To understand why Facebook experiences such a huge RTT hike in the 4G network, we analyze the servers connected, and find that there are no local Facebook servers hosted in Hong Kong for our tested 4G network, whereas local servers exist for the WiFi network.

As for the 188 nonresponsive connections, they either time out or fail for other reasons. Most of these connections come from WeChat and a news app by NetEase (the most popular news app in China). Our investigation discovers that WeChat's failures are due to their DNS misconfiguration for `hkminorshort.weixin.qq.com`. When WeChat queries this domain through the smartphone's DNS server (`8.8.8.8`), the latter therefore responds with `1.1.1.1` because of its wrong or unregistered DNS configuration. This finding was confirmed and acknowledged by Tencent, and they fixed the problem thereafter. On the other hand, NetEase's failures are due to their extremely large RTTs, causing 18.1% connections to exceed MopEye's three-second timeout setting.

4. CONCLUSION AND FUTURE WORK

We proposed a novel measurement app, called MopEye, to monitor per-app network performance on unrooted smartphones. We will deploy MopEye to Google Play soon for a large-scale measurement study.

Acknowledgements

We thank all four anonymous reviewers for their helpful comments. This work was partially supported by a grant (ref. no. ITS/073/12) from the Innovation Technology Fund in Hong Kong.

5. REFERENCES

[1] Y. Chen, E. Nahum, R. Gibbens, and D. Towsley. Measuring cellular networks: Characterizing 3G, 4G, and path diversity. In *UMass Amherst Technical Report: UM-CS-2012-022*, 2012.

[2] Configure and manage VPN connections programmatically in iOS 8. http://ramezanpour.net/post/2014/08/03/configure-and-manage-vpn-connections-programmatically-in-ios-8/.

[3] W. Li, R. Mok, D. Wu, and R. Chang. On the accuracy of smartphone-based mobile network measurement. In *Proc. IEEE INFOCOM*, 2015.

[4] MobiPerf on Google Play. https://play.google.com/store/apps/details?id=com.mobiperf.

[5] Netalyzr on Google Play. https://play.google.com/store/apps/details?id=edu.berkeley.icsi.netalyzr.android.

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

[7] N. 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.

[8] Speedtest.net on Google Play. https://play.google.com/store/apps/details?id=org.zwanoo.android.speedtest.

[9] VpnService | Android Developers. http://developer.android.com/reference/android/net/VpnService.html.