PePa Ping Dataset: Comprehensive Contextualization of Periodic Passive Ping in Wireless Networks

Diego Madariaga
diego@niclabs.cl
NIC Chile Research Labs
University of Chile

Lucas Torrealba
lucas@niclabs.cl
NIC Chile Research Labs
University of Chile

Javier Madariaga
javier@niclabs.cl
NIC Chile Research Labs
University of Chile

Javier Bustos-Jiménez
jbustos@niclabs.cl
NIC Chile Research Labs
University of Chile

Benjamin Bustos
bebustos@dcc.uchile.cl
Department of Computer Science
University of Chile

ABSTRACT

Among all Internet Quality of Service (QoS) indicators, Round-trip time (RTT), jitter and packet loss have been thoroughly studied due to their great impact on the overall network's performance and the Quality of Experience (QoE) perceived by the users. Considering that, we managed to generate a real-world dataset with a comprehensive contextualization of these important quality indicators by passively monitoring the network in user-space. To generate this dataset, we first developed a novel Periodic Passive Ping (PePa Ping) methodology for Android devices. Contrary to other works, PePa Ping periodically obtains RTT, jitter, and number of lost packets of all TCP connections. This passive approach relies on the implementation of a local VPN server residing inside the client device to manage all Internet traffic and obtain QoS information of the connections established. The collected QoS indicators are provided directly by the Linux kernel, and therefore, they are exceptionally close to real QoS values experienced by users' devices. Additionally, the PePa Ping application continuously measured other indicators related to each individual network flow, the state of the device, and the state of the Internet connection (either WiFi or Mobile). With all the collected information, each network flow can be precisely linked to a set of environmental data that provides a comprehensive contextualization of each individual connection.

CCS CONCEPTS

• **Networks** → **Network measurement**; **Mobile networks**; *Network performance analysis.*

KEYWORDS

dataset, crowdsourced measurements, wireless networks, network performance

MMSys 21, September 28-October 1, 2021, Istanbul, Turkey

ACM ISBN 978-1-4503-8434-6/21/09…$15.00
https://doi.org/10.1145/3458305.3478456

ACM Reference Format:
Diego Madariaga, Lucas Torrealba, Javier Madariaga, Javier Bustos-Jiménez, and Benjamin Bustos. 2021. PePa Ping Dataset: Comprehensive Contextualization of Periodic Passive Ping in Wireless Networks. In *12th ACM Multimedia Systems Conference (MMSys '21) (MMSys 21), September 28-October 1, 2021, Istanbul, Turkey.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3458305.3478456

1 INTRODUCTION

Among all factors that affect the network service quality, Round-trip time, jitter, and packet loss are three QoS indicators that have been thoroughly studied due to their great influence on the overall network's performance [17]. Both academic and engineering researchers have analyzed these QoS indicators for network management purposes, such as their use in TCP stacks to help optimize bandwidth usage [29].

In addition, RTT, jitter, and packet loss indicators have received a great deal of attention due to their huge impact on the user's quality of experience (QoE). Unfavorable network conditions may translate into high values of these indicators, which can be perceived by the user in the form of network delays or inability to establish connections. Their impact in QoE has already been studied in some scenarios such as content delivery networks [6], multiplayer gaming [3], HTTP video streaming [21], VoIP [14] and popular mobile services and apps [7].

In this paper, we present a dataset generated by a novel methodology for passive monitoring of all Internet traffic in mobile devices without the need of user intervention or root permissions to run properly. We are able to do this by taking advantage of Android's APIs and Linux kernel's TCP facilities. In this manner, we can periodically run our system to take passive QoS measurements from all TCP connections established, such as RTT, jitter, and number of lost packets. This methodology, presented in Section 3, allows us to obtain empirical QoS information from the individual connections that users are employing to access the Internet in completely normal usage scenarios. Section 4 presents a description of how we used our passive monitoring methodology to generate a valuable real-world dataset that precisely combines network flow information and contextual information.

Our paper presents two clear contributions. Firstly, to the best of the authors' knowledge, this is the first research work focused on

mobile devices that presents a methodology to obtain QoS information from real Internet usage by periodically taking non-intrusive passive measurements in user-space. Secondly, this study presents a valuable dataset that allows a detailed contextualization of Internet connections measured in the wild. The presented dataset has great significance since it can be used to study network usage and network performance in a detailed manner.

2 RELATED WORK

2.1 Network Ping Indicators

The most common tool for measuring RTT, jitter and packet loss is the Ping networking tool. The Ping utility sends ICMP (Internet Control Message Protocol) packets of type `echo request` and awaits for ICMP packets of type `echo reply`. There are two main drawbacks when using this tool for measuring a device's received QoS. i) It does not provide precise information on the received QoS because traffic control mechanisms have different rules for ICMP and TCP/UDP traffic [29, 31] and ii) as most active measurements, Ping only measures the quality of a single connection and not the whole state of the user's Internet connectivity.

2.2 Passive Ping Approaches

As a way to face these problems, researchers and network operators have adopted passive Ping approaches, enabling the analysis of every data flow without inducing a traffic overload that could affect their measurements. The following methods are some of the known passive approaches for estimating TCP RTT of a device's Internet connections: i) capturing the timestamp of TCP handshake packets [13], ii) timing TCP packets in a remote measurement machine [19], iii) analyzing the `timestamp` option in the TCP header[1] using the mechanism described by Jacobson et al. [12] and iv) capturing the timestamp of TCP handshake packets in a remote monitoring probe [27]. Unlike these approaches, the periodic passive `ping` method presented in this work (PePa Ping) is able to obtain RTT, jitter and number of lost packets from within the device's system by taking advantage of Linux kernel's TCP facilities. Our method does not need any user intervention, root permission or remote measurement machine, making it easy to distribute to real users.

2.3 Traffic Monitoring Through Local VPN

Even though there are many projects related to measuring QoS on mobile networks, to the author's knowledge, this is the first to use a local VPN approach. We call local or internal VPN to a mounted or simulated VPN server inside the same client device, acting like a man-in-the-middle proxy. Different from our work, most of the projects that monitor mobile networks using an internal VPN are focused on malware and privacy leakage detection [26, 28]. PrivacyGuard [28] is a VPN-based Android platform that detects information leakage by intercepting applications' network traffic using a local VPN. While it uses the same approach as this work, PrivacyGuard needs to be able to read all traffic information in plain text even if it comes encrypted using TLS protocol. This means doing TLS interception and packet inspection. PePa Ping does not incorporate any overhead of such type due to its passive nature. Our work only uses the VPN to retransmit the packets through its own sockets, which allows to easily measure RTT, jitter and number of lost packets. On the same line there is Haystack [26], a VPN-based Android platform that analyzes mobile Internet traffic, locally and in user-space, in order to characterize mobile traffic and detect privacy risks. Our work uses the same user-space VPN approach as Haystack, with its main differences being that PePa Ping implements its sockets in C++, which allows for it to retrieve QoS-related metrics for each flow, and its periodic behavior, showing a lower impact on the device's battery in comparison to a continuous monitoring system.

2.4 Multidimensional QoS Analyses

State-of-the-art works on characterizing mobile QoS usually perform multidimensional analysis to describe network performance. Several authors [5, 10, 20, 23] looked at end-to-end performance by taking active measurements in user-space and correlating multiple QoS indicators with user's contextual information. Our work differs from theirs as we perform passive measurements on real user traffic. In many works [2, 16, 27, 30], QoS indicators of TCP flows were obtained from inside mobile carriers using passive monitoring probes. With this approach they were not able to collect user's contextual information and, moreover, the estimated QoS indicators may not match what the user perceived. Indeed, apart from ours, no comprehensive work was dedicated to measure the QoS of all existing Internet connections in mobile devices by only taking passive measurements in user-space.

3 MEASUREMENT METHODOLOGY

To generate the presented dataset, we developed PePa Ping (*Periodic Passive Ping*), an Android framework capable of taking periodic *Ping-like* measurements (RTT, jitter, and number of lost packets) through passive monitoring of active connections without injecting any new packet into the network, contrary to standard Ping and its ICMP packets. The above is achieved by taking advantage of Android WorkManager API, Android VpnService, and querying the OS directly for QoS information.

In the following subsections, we will explain the methodology behind PePa Ping, which is represented by its three main characteristics: being periodic, passive, and *Ping-like*.

3.1 Periodic Behavior

One of the main concerns when developing a monitoring system for mobile devices is the overhead it may produce in CPU and battery consumption. This is the reason for choosing a periodic monitoring system over a continuous one. With enough periodicity, we can obtain a good representation of the network's traffic without overloading the device.

WorkManager is an API provided by Android since 2018. This API makes it easy to schedule deferrable, asynchronous tasks that are expected to run even if the app exits or the device restarts[2]. With this API, we schedule our monitoring system to start running for 1 minute every 15 minutes. We chose 15-minute intervals because it is the minimum duration that WorkManager allows for periodic tasks to be scheduled without them interfering with the device's

[1] http://www.pollere.net/pping.html

[2] https://developer.android.com/topic/libraries/architecture/workmanager

battery optimization system. Regarding the monitoring time of our system, we chose 1-minute slots as an attempt to impact the normal operation of users' devices minimally. Nevertheless, given the regularity of both human mobility patterns [24] and Internet usage patterns [32], the selected methodology is expected to be representative of the whole Internet traffic generated by users.

3.2 Passive Monitoring

According to RFC 7799 [22] passive methods have these characteristics: they are i) based solely on observations of an undisturbed and unmodified packet stream of interest, ii) dependent on the existence of one or more packet streams to supply the stream of interest and iii) dependent on the presence of the packet stream of interest at one or more designated Observation Points. While trying our best to follow these rules, we designed a system that uses Android's VpnService to implement a local VPN server written in C++. Next, we will explain this in detail.

3.2.1 VpnService. PePa Ping implements an Android VpnService in order to gain packet-level access without requiring root privileges. According to Android's documentation, VpnService is a base class for applications to extend and build their own VPN solutions[3]. VpnService gives a file descriptor `fd` to the application for it to read and write on. Each read from `fd` retrieves an outgoing IP packet which was routed to the interface. Each write to `fd` injects an incoming IP packet just like it was received from the interface. A typical VPN-client application completes the VPN connection by processing and exchanging packets with the remote server over a tunnel. In our case, we complete the VPN connection by handing the packets over to our local VPN server, which handles the traffic processing.

3.2.2 Local VPN Server (Packet Forwarder). Our local VPN server is implemented in C++. It is able to run within the application's Java code by using the Java Native Interface (JNI) framework. The local server receives the outgoing IP packets through the VPN's `fd` and sends only its payload, without any headers, through an application-level socket of type `SOCK_DGRAM` for UDP or type `SOCK_STREAM` for TCP. Given that the VPN's `fd` only receives IP packets, but reading from an application-level socket returns a TCP/UDP packet's payload, manually handling its headers is necessary before injecting them into the `fd`. Creating artificial headers differs for UDP and TCP. In both cases, we have got to calculate the IP checksum. The custom UDP header is simple because of its connectionless nature, and the main work consists of calculating the packet length and the UDP checksum. TCP is more complex because it involves keeping track of handshake packets and acknowledgment and sequence numbers of data packets. This procedure is explained in further detail in Razaghpanah et al. [26].

The local VPN keeps track of the current connections using two C++ unordered maps. These maps store TCP and UDP connections represented by a C++ class called `VpnConnection`, with the map unique-key being the concatenation of the origin's port, destination IP, and destination port. The `VpnConnection` object stores the socket that communicates our local VPN with the destination address. Furthermore, we implement a polling method using the system call `epoll`, used to monitor multiple file descriptors and generate an event signaling that I/O is possible on any of them. Having a polling method is necessary for high-performance networking applications with non-blocking socket I/O.

Even though other authors implement this local VPN approach [26, 28], they implement the packet forwarding mechanism in Java. We chose to implement it in C++ because it grants us access to specific connection-level information, as mentioned in Section 3.3.

3.3 Ping Information

Our system registers the open and close time of every TCP and UDP socket using the POSIX function `gettimeofday()` that provides current time with microsecond (μs) precision. In addition, when a TCP socket (`SOCK_STREAM` type) is closed, we call `getsockopt()` function with option `TCP_INFO` that retrieves a `struct tcp_info` variable. This variable gives us access to information about the closed TCP connection, directly from the Linux kernel. Using this method, we gain access to every TCP connection's `rtt`, `rtt_var` and `lost_packets` variables, similar to the information obtained by the Ping tool. The `rtt` and `rtt_var` variables are estimations of RTT mean and RTT standard deviation (commonly referred to as jitter), which are used for TCP congestion control [4, 25]. The Linux kernel calculates these estimations using Jacobson's algorithm [1, 11].

It is important to clarify that even when the network performance perceived by the users may be affected by the overhead of our local VPN server, there's no overhead in `rtt`, `rtt_var` or `lost_packets` measurements, given that they are taken from sockets connected directly from our local VPN server to the destination address. Therefore, these measurements characterize the real QoS of the established connections.

As Android is based on a modified version of the Linux kernel, it would be possible to obtain these low-level quality indicators using other client-side monitoring approaches such as *Wireshark* and *tcpdump*. However, these methods require to cross-compile the *libpcap* library for Android. Moreover, *libpcac* must be run with superuser privileges to work properly, and therefore, these methods can only be executed in rooted Android phones. Consequently, these approaches are not suitable for a crowdsourcing scenario.

4 DATASET GENERATION

To test our measurement methodology with real users and to generate the presented dataset, we built an Android application that incorporates the PePa Ping framework described in Section 3. In addition, the application collects a comprehensive set of network-related contextual information in parallel with each 1-min execution of the PePa Ping methodology. Finally, we also developed a database server to store all collected data in a PostgreSQL database. User devices sent their data to this server through secure HTTP connections (HTTPS). The dataset and a complete description of all SQL tables in the database can be found on the project repository[4].

The information stored in the PostgreSQL database can be divided into three different groups: Network flow information, Environmental information, and External information.

[3] https://developer.android.com/reference/android/net/VpnService

[4] https://github.com/niclabs/pepa-ping-mmsys21

4.1 Network Flow Information

As mentioned in Section 3.2, we collected information from all TCP and UDP connections throughout our system runs (PePa Ping methodology). By using our C++ VPN approach, we collected the following information for all network flow: destination IP address and port, network protocol, start time and end time (with microsecond precision), number of bytes transmitted and received, and package name of the Android application that established the connection. In addition, for all TCP connections, we collected the following information accessing to the Linux kernel `tcp_info` structure: Average round-trip time (RTT), minimum RTT, RTT variance (jitter), and number of lost packets. The captured information was stored in the database in the table *pepa_ping*. Finally, each network flow stored in the *pepa_ping* table includes a reference to the environmental information associated with a specific 1-min execution of the measurement system. Therefore, all rows in *pepa_ping* that reference the same environmental information correspond to network flows established during the same 1-min execution of the PePa Ping methodology.

4.2 Environmental Information

Our Android application continuously collected contextual information for the duration of each 1-min execution of PePa Ping. This information allows for a better understanding of the conditions in which the Internet connections of a specific 1-min execution were generated. The contextual information of each 1-min execution is composed by five SQL tables: *environmental_data*, *connection*, *cellular*, *wifi*, and *ram*.

- Firstly, the SQL table *environmental_data* contains the information that remains invariable throughout the whole minute of measurement: timestamp of the start of the 1-min execution, identifier of the user device, and identifier of the network operator that issued the device's SIM card. The remaining four SQL tables contain environmental information that varies throughout the execution of the measurement system. Each row in these four tables references to a specific row in *environmental_data* table.
- The table *connection* contains information related to Internet connectivity changes between Wi-Fi and Mobile Data: timestamp of the measurement, type of connection (Wi-Fi or Mobile Data), and number of the autonomous system to which the sending router belongs. The connectivity changes are tracked by registering an Android broadcast receiver for the intent `android.net.conn.CONNECTIVITY_CHANGE`.
- The table *cellular* contains information related to the cellular network quality throughout the 1-min execution of the measurement system. It is important to note that the information collected varies according to the mobile network technology being used. We used the Android *PhoneStateListener* class for monitoring changes to observed cell info and for monitoring changes to the network signal strength. Each row in the *cellular* table includes a timestamp of the measurement, information about the cell type (by inspecting the corresponding Android CellInfo object), information about the network technology being used (using the `getNetworkType()` function), and the identifier of the antenna used to access the mobile network. Additionally, there are different variables related to the signal quality depending on the technology, such as Received Signal Strength Indicator (RSSI), Channel Quality Indicator (CQI), Reference Signals Received Power (RSRP), Reference Signal Received Quality (RSRQ), Reference Signal Signal To Noise Radio (RSSNR), Received Signal Code Power (RSCP), bit error rate, among others. It is important to mention that an execution of the PePa Ping methodology can include a set of *cellular* entries even if the device has not been using the mobile network to access the Internet (for example, if using Wi-Fi). The above is because the cell phone is still connected to the mobile network.
- The table *wifi* contains information about the Wi-Fi quality throughout the 1-min execution of the measurement system. Each row of this table includes the *timestamp* of the measurement, the Received Signal Strength Indicator, and the Wi-Fi frequency. The connectivity changes are tracked by registering an Android broadcast receiver for the intent `WifiManager.RSSI_CHANGED_ACTION`.
- Finally, the table *ram* contains information about the RAM usage throughout the 1-min execution of the measurement system. Each row of this table includes the timestamp of the measurement, the total amount of RAM, and the available free RAM. The changes in the RAM usage are tracked by parsing the `/proc/meminfo` virtual file every 5 seconds.

Figure 1 presents an example of the information collected by our measurement application in a 1-min execution. The figure shows different Android applications that establish connections of different durations throughout the 60 seconds of measurement. In addition, a variety of contextual information is also displayed as a set of time series accompanying the connections (signal quality information and % of used RAM). These time series provide a comprehensive contextualization of the 1-min execution, showing all the temporal changes in detail. Moreover, these time series provide a precise contextualization of every single connection established.

Similarly, Figure 2 shows another example of an execution of our measurement methodology. The figure shows different applications establishing Internet connections using the mobile network. Some of these connections experienced a vertical handover between the mobile network technologies LTE and WCDMA. It is important to notice that we can precisely determine which connections were affected by the handover events and which were not.

4.3 External Information

In addition to the information collected directly by our Android application (network flow information and environmental information), we also collected some external information related to the destination IP and port of the network flow information obtained by the application. This external information is useful to characterize the services accessed by the different Android applications and gain further insights into network flows. This information is composed by four SQL tables: *nmap_service*, *autonomous_system*, *autonomous_system_info server_information* and *domain_tag*.

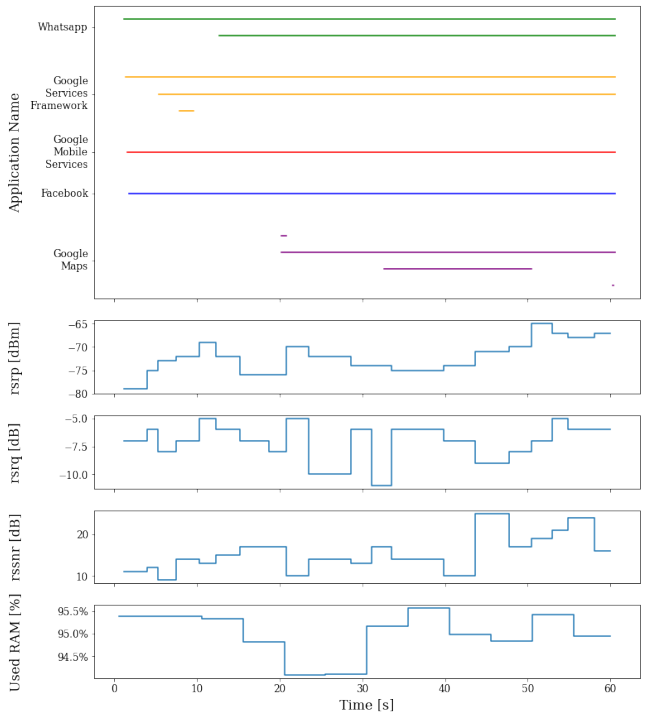

Figure 1: Example of the information collected during a execution of our PePa Ping methodology.

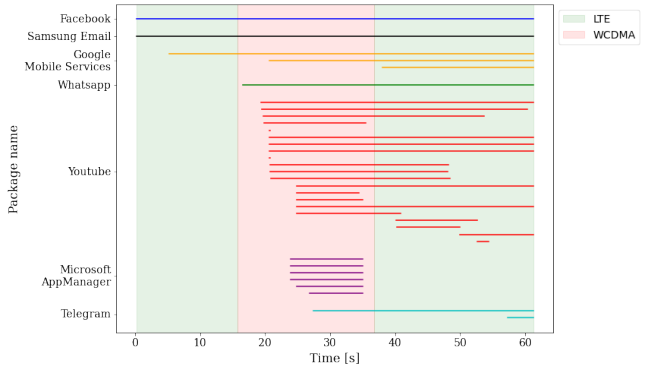

Figure 2: Example of an execution of our PePa Ping methodology with the presence of vertical handovers.

- The table *nmap_service* stores information about the service running on each pair `dst_ip:dst_port` from the collected dataset. This information was obtained by using nmap tool[5].
- The table *autonomous_system* stores information about the autonomous system number associated with each destination IP address. This information was obtained by using an IP WHOIS Lookups tool.
- The table *autonomous_system_info* stores information about each autonomous system number present in the table *autonomous_system*, as name and country. This information was obtained by performing IP WHOIS Lookups
- The table *server_information* stores information about entity that manages a specific IP address. This information was obtained by inspecting TLS/SSL certificates' meta-data and performing reverse DNS resolutions.
- Finally, the *domain_tag* table stores information about the categorization of different server names (or common names). This information was obtained by using the OpenDNS Community's Domain Tagging[6].

To generate this dataset, we conducted a recruitment campaign and distributed our crowdsourcing application to people in the city of Santiago, Chile. From January 7 to March 17 in 2021, network data was collected from 137 real users who voluntarily accepted installing the application on their personal devices. In total, the application registered information about ~220,000 executions of the 1-min measurement system. Thus, the monitoring system collected information about ~3,200,000 Internet traffic flows (~2,300,000 TCP connections and ~900,000 UDP connections). These connections were established by 1255 different Android applications to ~30,000 different IP addresses. As the users were recruited following a convenience sampling process, the Android applications present in the collected dataset are influenced by the over-representation of university students in the population sample. However, there is a great diversity among the network usage during the executions of our system. This diversity is evidenced in the number of Internet connections established during the ~220,000 1-min executions measured (Figure 3). The figure presents the cumulative distribution function of the number of connections established in an execution of the PePa Ping methodology. Additionally, Figure 4 shows the number of executions of our 1-min measurement system per user, evidencing how much data was collected by each participant.

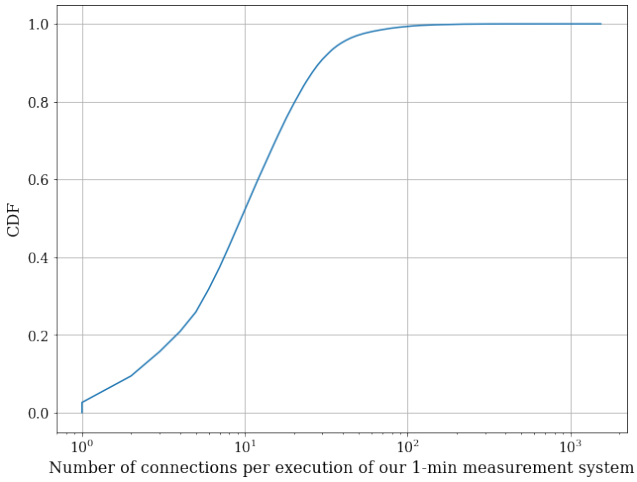

Figure 3: CDF of the number of connections established per execution of our PePa Ping methodology.

[5]https://nmap.org/

[6]https://community.opendns.com/domaintagging/

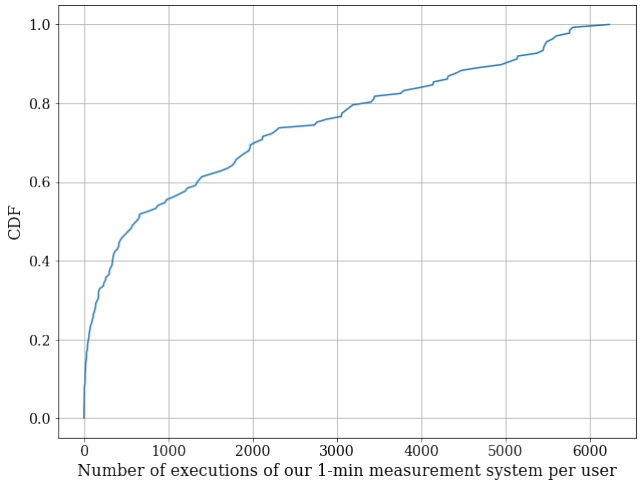

Figure 4: CDF of the number of executions of our PePa Ping methodology per user.

Ethical Considerations. In order to keep the ethical considerations of network measurements and management of user data, this study was conducted ensuring that full permissions are sought from and provided by users of the measurement app (PePa Ping app [15]). After being shown a full description of PePa Ping project (the collected information, how data will be used and who will access and use the dataset), we asked participants to grant us permission to use their collected information for project-related research activities before installing the crowdsourcing application. Collected measurements in the presented dataset were appropriately anonymized to ensure the privacy of participants. Furthermore, this study has been approved by the ethics committee of the University of Chile.

5 CONCLUSIONS

In this paper, we presented the dataset collected by our PePa Ping methodology, which is publicly available on the project repository. The generated database contains valuable information about network usage, overall empirical QoS of TCP flows, and a comprehensive set of mobile user's contextual information. Thus, we provide a useful source that can be of great importance to a variety of studies related to network usage behavior and network performance.

Among its possible use cases, the information about network usage included in the dataset can be used to analyze the adoption of network protocols such as QUIC [18]. Also, the dataset provides a unique source to study the effect of handovers on the performance of users' connections in the wild. The contextual information collected throughout each 1-min execution of the PePa Ping methodology allows precise identification of both horizontal handovers (between the same network technology) and vertical handovers (between different network technologies). A better understanding of these effects is essential since previous works have already discussed the negative impact of handovers on the performance of mobile networks [8, 27], and the sudden increase of RTT values after a handover [9]. As a very first approach, Figure 5 and Figure 6 evidence almost no effect of horizontal handovers on the quality of 4G LTE connections, unlike the case of 3G connections that exhibit a higher impact. Finally, there is a vast amount of research and experimentation left to do to better understand the relation between Internet QoS and the network state. Thus, the presented dataset can be helpful to study these phenomena and propose novel models to relevant problems for the community, such as round-trip time and throughput prediction.

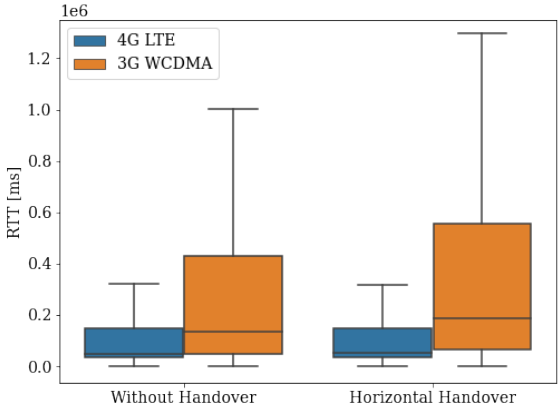

Figure 5: Boxplot chart summarizing the impact on RTT of horizontal handover among different network technologies.

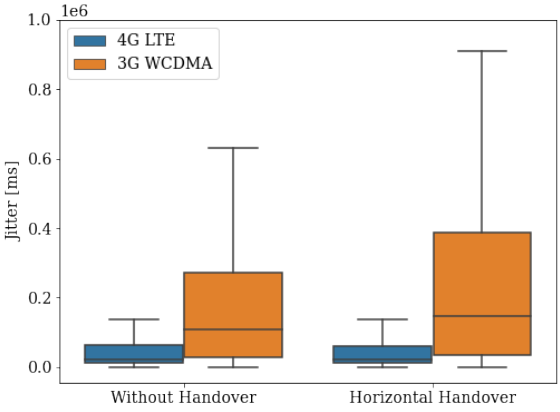

Figure 6: Boxplot chart summarizing the impact on jitter of horizontal handover among different network technologies.

ACKNOWLEDGMENTS

This work was partially funded by the National Agency for Research and Development (ANID) / Scholarship Program / Doctorado Nacional 2019 - 21190450 and by the Millennium Institute for Foundational Research on Data (IMFD).

REFERENCES

[1] Somaya Arianfar. 2012. TCP's congestion control implementation in Linux kernel. In *Proceedings of Seminar on Network Protocols in Operating Systems*. 16.

[2] Peter Benko, Gabor Malicsko, and Andras Veres. 2004. A large-scale, passive analysis of end-to-end TCP performance over GPRS. In *IEEE INFOCOM 2004*, Vol. 3. IEEE, 1882–1892.

[3] Anastasiia Beznosyk, Peter Quax, Karin Coninx, and Wim Lamotte. 2011. Influence of network delay and jitter on cooperation in multiplayer games. In *Proceedings of the 10th international conference on virtual reality continuum and its applications in industry*. 351–354.

[4] Robert Braden. 1989. RFC1122: Requirements for Internet hosts-communication layers.

[5] Edy Budiman and Oki Wicaksono. 2016. Measuring quality of service for mobile internet services. In *2016 2nd International Conference on Science in Information Technology (ICSITech)*. IEEE, 300–305.

[6] Pedro Casas, Alessandro D'Alconzo, Pierdomenico Fiadino, Arian Bär, Alessandro Finamore, and Tanja Zseby. 2014. When YouTube does not work—Analysis of QoE-relevant degradation in Google CDN traffic. *IEEE Transactions on Network and Service Management* 11, 4 (2014), 441–457.

[7] Pedro Casas, Michael Seufert, Florian Wamser, Bruno Gardlo, Andreas Sackl, and Raimund Schatz. 2016. Next to you: Monitoring quality of experience in cellular networks from the end-devices. *IEEE Transactions on Network and Service Management* 13, 2 (2016), 181–196.

[8] Andrei Gurtov and Jouni Korhonen. 2004. Effect of vertical handovers on performance of TCP-friendly rate control. *ACM SIGMOBILE Mobile Computing and Communications Review* 8, 3 (2004), 73–87.

[9] Andrei Gurtov and Reiner Ludwig. 2003. Responding to spurious timeouts in TCP. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*, Vol. 3. IEEE, 2312–2322.

[10] Junxian Huang, Qiang Xu, Birjodh Tiwana, Z Morley Mao, Ming Zhang, and Paramvir Bahl. 2010. Anatomizing application performance differences on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*. 165–178.

[11] Van Jacobson. 1988. Congestion avoidance and control. *ACM SIGCOMM computer communication review* 18, 4 (1988), 314–329.

[12] Van Jacobson, Robert Braden, and Dave Borman. 1992. RFC1323: TCP extensions for high performance.

[13] Hao Jiang and Constantinos Dovrolis. 2002. Passive estimation of TCP round-trip times. *ACM SIGCOMM Computer Communication Review* 32, 3 (2002), 75–88.

[14] Mansour J Karam and Fouad A Tobagi. 2002. Analysis of delay and delay jitter of voice traffic in the Internet. *Computer Networks* 40, 6 (2002), 711–726.

[15] NIC Chile Research Labs. 2020. *PePa Ping: Measuring QoS for mobile Internet*. https://niclabs.cl/pepa/

[16] Markus Laner, Philipp Svoboda, Eduard Hasenleithner, and Markus Rupp. 2011. Dissecting 3G uplink delay by measuring in an operational HSPA network. In *International Conference on Passive and Active Network Measurement*. Springer, 52–61.

[17] Hyo-Jin Lee, Myung-Sup Kim, James W Hong, and Gil-Haeng Lee. 2002. QoS parameters to network performance metrics mapping for SLA monitoring. *KNOM Review* 5, 2 (2002), 42–53.

[18] Diego Madariaga, Lucas Torrealba, Javier Madariaga, Javiera Bermúdez, and Javier Bustos-Jiménez. 2020. Analyzing the Adoption of QUIC From a Mobile Development Perspective. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*. 35–41.

[19] H Martin, Anthony McGregor, and J Cleary. 2000. Analysis of internet delay times. In *Proceedings of Passive and Active Measurment Workshop (PAM)*.

[20] Lars M Mikkelsen, Steffen R Thomsen, Michael S Pedersen, and Tatiana K Madsen. 2014. NetMap-creating a map of application layer qos metrics of mobile networks using crowd sourcing. In *International Conference on Next Generation Wired/Wireless Networking*. Springer, 544–555.

[21] Ricky KP Mok, Edmond WW Chan, and Rocky KC Chang. 2011. Measuring the quality of experience of HTTP video streaming. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*. IEEE, 485–492.

[22] A Morton. 2016. Active and passive metrics and methods (with hybrid types in-between). *Internet Engineering Task Force* (2016).

[23] Ashkan Nikravesh, David R Choffnes, Ethan Katz-Bassett, Z Morley Mao, and Matt Welsh. 2014. Mobile network performance from user devices: A longitudinal, multidimensional analysis. In *International Conference on Passive and Active Network Measurement*. Springer, 12–22.

[24] Eduardo Mucelli Rezende Oliveira, Aline Carneiro Viana, Carlos Sarraute, Jorge Brea, and Ignacio Alvarez-Hamelin. 2016. On the regularity of human mobility. *Pervasive and Mobile Computing* 33 (2016), 73–90.

[25] Jon Postel et al. 1981. Transmission control protocol RFC 793.

[26] Abbas Razaghpanah, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Christian Kreibich, Phillipa Gill, Mark Allman, and Vern Paxson. 2015. Haystack: In situ mobile traffic analysis in user space. *arXiv preprint arXiv:1510.01419* (2015), 1–13.

[27] Peter Romirer-Maierhofer, Fabio Ricciato, Alessandro D'Alconzo, Robert Franzan, and Wolfgang Karner. 2009. Network-wide measurements of TCP RTT in 3G. In *International Workshop on Traffic Monitoring and Analysis*. Springer, 17–25.

[28] Yihang Song and Urs Hengartner. 2015. Privacyguard: A vpn-based platform to detect information leakage on android devices. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. 15–26.

[29] Stephen D Strowes. 2013. Passively measuring TCP round-trip times. *Commun. ACM* 56, 10 (2013), 57–64.

[30] Francesco Vacirca, Fabio Ricciato, and René Pilz. 2005. Large-scale RTT measurements from an operational UMTS/GPRS network. In *First International Conference on Wireless Internet (WICON'05)*. IEEE, 190–197.

[31] Li Wenwei, Zhang Dafang, Yang Jinmin, and Xie Gaogang. 2007. On evaluating the differences of TCP and ICMP in network measurement. *Computer Communications* 30, 2 (2007), 428–439.

[32] Fengli Xu, Yuyun Lin, Jiaxin Huang, Di Wu, Hongzhi Shi, Jeungeun Song, and Yong Li. 2016. Big data driven mobile traffic understanding and forecasting: A time series approach. *IEEE transactions on services computing* 9, 5 (2016), 796–805.