Voice-activated applications and Multipath TCP: A good match?

Viet-Hoang Tran
ICTEAM, UCLouvain
Louvain-la-Neuve, Belgium
hoang.tran@uclouvain.be

Hajime Tazaki
IIJ Research Laboratory
Tokyo, Japan
tazaki@iij.ad.jp

Quentin De Coninck
ICTEAM, UCLouvain
Louvain-la-Neuve, Belgium
quentin.deconinck@uclouvain.be

Olivier Bonaventure
ICTEAM, UCLouvain
Louvain-la-Neuve, Belgium
olivier.bonaventure@uclouvain.be

***Abstract*—Voice is progressively becoming a popular way to interact with mobile devices such as smartphones or connected cars. Most of the current deployments depend on cloud services to recognize the user's commands. For this reason, voice-controlled applications have stringent requirements in terms of delay or availability. On the other hand, many of the devices using such applications are attached to several wireless networks. On iPhones, Multipath TCP made voice-enabled applications useable while users move from cellular to WiFi.**

In this paper, we leverage the MONROE platform to analyze the performance of Multipath TCP for voice-activated applications. For this, we port the Multipath TCP Linux kernel code into the Linux Kernel Library so that it can run as a regular application. We extend `iperf3` to emulate voice-activated applications and carry out measurement campaigns. Our measurements show that Multipath TCP brings clear benefits for users attached to two networks.

***Index Terms*—multipath, MPTCP, interactive traffic, voice-recognition, low-latency, measurement**

I. Introduction

Voice recognition has a long history, but the recent developments in deep learning, the improvement of hardware capability and the pervasiveness of the Internet have enabled a new generation of cloud-based voice recognition platforms, e.g. Apple Siri, Google Speech Recognition, Microsoft Cortana, Amazon Alexa. These platforms have enabled or potentially will enable several use cases in various activities of our lives: voice-based search or commands, hands-free infotainment control at home and in connected cars [1], automatic customer support [2], voice-based path guides for visually impaired people [3], etc. These systems do not require a high volume of network traffic, but they do present very specific networking requirements: high availability, low latency and energy awareness. These applications typically send voice samples to cloud servers that return the recognised text.

Since 2013, Siri, the voice-recognition and virtual assistant application on iOS, has used Multipath TCP (MPTCP) [4] by default to communicate with the Apple servers [5]. This is today the largest commercial deployment of Multipath TCP [6]. The large deployment of Siri makes it a baseline for other voice-enabled applications. In this paper, we would like to answer two questions: *(1) What are the benefits of using MPTCP for voice-recognition traffic?* and *(2) What are the factors that impact the performance of MPTCP?*

A first possible approach could be conducting passive measurement on the real Siri traffic at some vantage points such as university campus WiFi networks, or on mobile operator gateways. However, this approach is incomplete since Multipath TCP can use different paths and it is difficult to passively collect all the packets sent by a smartphone over WiFi and cellular.

Another approach is to leverage existing mobile measurement platforms to deploy simplified voice-activated applications and conduct active measurements. The challenge now is to have a mature Multipath TCP stack on the mobile nodes used on those platforms. In this paper, we present a methodology to conduct Multipath TCP measurements on a mobile broadband platform and leverage the Linux Kernel Library [7] to quickly deploy Multipath TCP. We use this methodology to collect sample measurements to demonstrate its benefits by answering the two questions above.

This paper is organised as follows. Section II provides a background on the voice-recognition traffic and our observations on recent Siri traffic. Section III describes the MONROE platform, the challenges to run Multipath TCP on this platform and our approach. The measurement methods and procedures are depicted in detail in Section IV. Section V presents the results of two measurement campaigns and the Section VI concludes our paper.

II. Background

In this section, we briefly describe the behaviour of voice-recognition applications such as Siri and Multipath TCP.

A. Voice-recognition Traffic

Traditionally, unreliable protocols like UDP are used to transport interactive voice traffic to avoid the additional latency induced by reliable transport protocols. However, most of the deployed cloud-based voice-recognition systems use TCP [8] or QUIC[9] - which is on top of UDP but has added reliability - as their transport protocols. This is likely because the responses from servers are typically textual or binary messages, which need to be transferred reliably.

The network traffic generated by Siri has been preliminary analysed by several works [10], [11], [12], [13].

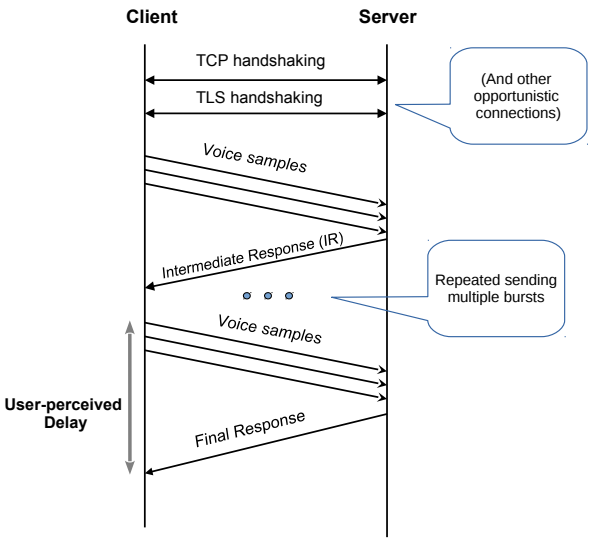


Fig. 1: The observed Siri traffic pattern

Our observations on Siri traffic: In order to have up-to-date information on the current Siri communication behavior, we have captured Siri traffic on an iPhone running iOS 11.1 (Fig. 1). The tests included a series of real-life questions, a current weather query, and a one-off Google search. Below are some of our observations, which are used for traffic emulation in our measurements.

Siri uses TLS 1.2 for encrypting the exchanged data. As an interesting side note, shortly after the beginning of the Siri connection, we also observe that other TCP connections are established towards other servers for other Apple services, e.g. weather forecast, notification update, new system update check. These connections are probably created opportunistically to save the energy consumed due to activating network interfaces. Except in the TLS handshaking phase, the clients send data in bursts of several small TCP segments, which likely contain the encoded voice samples. The length of each segment lies in the range of 50-500 Bytes and always has the `PSH` flag set, showing that the `TCP NO_DELAY` socket option has been used. After each burst, the server sends back a small response. It is believed that Siri uses HTTPS as the application protocol for the transaction [10], in which the voice samples are included in HTTP POST messages. Thus, the small responses that we observe are probably the HTTP `100 Continue` informational status messages. When all voice samples have been received and processed, the server sends back its final response. After that, the connection is not closed immediately by the client nor the server, but is maintained persistently for a long time. This process is illustrated in Figure 1.

B. Multipath TCP

Multipath TCP [4] is a TCP extension whose specification was published in 2013. It enables hosts to exchange packets over different paths to improve performance or reliability. Multipath TCP allows a mobile application to start a connection over the WiFi interface and then automatically switch to the cellular interface when the quality of the WiFi network decreases [14]. This ability to efficiently support handovers is the main reason why Apple has been using Multipath TCP since 2013 [6].

Multipath TCP is described in detail in [4] and in [15]. In a nutshell, Multipath TCP allows to use different paths by establishing one TCP connection, called subflow over each path. The subflows that compose a Multipath TCP connection are not static, they can be established and released at any time during the lifetime of a connection. Multipath TCP allows data to be sent over any path and data sent over one path could be retransmitted over another path if the former fails.

A Multipath TCP implementation typically includes two types of algorithms to control the utilisation of the different paths. The path manager controls the establishment of subflows. A simple path manager could simply create all subflows at connection establishment time. Advanced path managers [16] can delay the establishment of subflows and only create them when the primary path fails [14] or when its throughput is too low. The packet scheduler is another important algorithm found in any Multipath TCP implementation [17]. It selects over which path each data is transmitted. Several schedulers exist. The simplest one is the round-robin scheduler that uses all established subflows. The default scheduler in the Multipath TCP implementation in the Linux kernel [18] prefers the subflow with the lowest round-trip-time [17].

III. MPTCP Measurements: Challenges and Approach

We use the MONROE platform [19] for our measurements since it supports various multi-homed wireless nodes. To realise our MPTCP measurements on the MONROE platform, we have to overcome a technical challenge. Since the platform only allows experimenters to run their tests inside Docker container [20], we cannot run the experimental MPTCP stack in the Linux kernel directly. To run a Linux MPTCP implementation in user-land, we extend the Linux kernel library to provide Linux MPTCP to the application.

A. The MONROE Platform

MONROE [19] is a large multi-homed mobile-broadband measurement platform. It provides the experimenters the access to two kinds of nodes: stationary nodes and mobile nodes (which are placed on trains, trucks, or buses). Detailed descriptions of the platform have been presented earlier [21], [22]. For example, [22] presents simple but extensive download-/upload measurements on this platform. In our experiments, we observed - on more than half of the nodes - that the `MP_CAPABLE` option [4] used by Multipath TCP to establish connections was removed from the SYN packet on port 80 towards an external server. However, the MPTCP connections on other port ranges (5201 to 5300) do not suffer from this problem. This suggests that HTTP transparent proxies are used

in the cellular networks attached to these nodes, confirming the observation in [22].

As the MONROE platform is based on a container technology, introducing MPTCP to an application inside a container requires kernel upgrades, which is challenging to deploy on the MONROE nodes. We solve this problem by using the Linux kernel library to allow an application to use a custom MPTCP-enabled network stack without requiring any kernel change on the host.

B. Linux Kernel Library overview

The Linux kernel library (LKL) [7] is essentially a library that enables users to use custom Linux kernel code directly inside applications. Unlike a typical build of Linux kernel source tree which produces a bootable image, a build of LKL generates a set of library files which contain the Linux kernel but live in userspace. An application can link the LKL library in order to call alternate system calls (instead of the ones of the host kernel) implemented in LKL. We leverage this feature to use the Multipath TCP network stack for our test application.

After traversing packet processing in the LKL system calls (completely operated inside userspace), a packet goes through a virtualized device driver composed of Linux kernel code, which is a `virtio` driver implementation, and destined to a `virtio` device of LKL to be transmitted to the outside. LKL supports a series of `virtio` devices: raw socket, tap device, Virtual Distributed Ethernet (VDE) [23], Intel Data Plane Development Kit (DPDK) [24], etc. With those devices, the incoming and outgoing packets processed by the network stack of LKL do not need to pass through the stack of the host operating system.

IV. Measurement Design

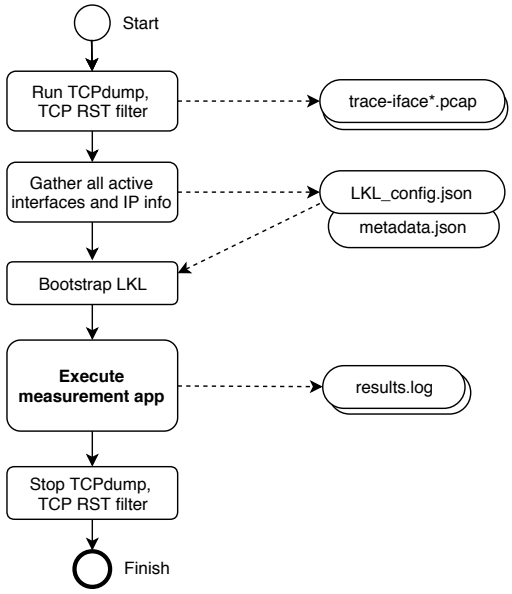


Fig. 2: General Experiment Procedure with LKL

The MONROE platform consists of not only stationary nodes which are located in four European countries (Norway, Sweden, Italy and Spain) but also mobile nodes which are set up on trains (Norway), buses (Sweden, Italy), and trucks (Italy). For our experiments, we selected 28 stationary nodes and 40 mobbile nodes - each of them has two cellular interfaces which are connected to different mobile operators.

To reduce the location bias due to the utilisation of a single server, we set up two servers: one on our university campus in Belgium and the other one in Tokyo, Japan. However, due to space limitations, we only present the results with Belgian server in this paper. Both clients and servers run version 0.93 of the Multipath TCP implementation in the Linux kernel [18]. On client, we need to merge the source code of LKL and Linux Multipath TCP and built them as a library to the client application [1]. We also use the enhanced socket API [25] to only use the wireless interfaces for creating the subflows. Several coupled congestion controls have been proposed to maintain the fairness with regular TCP. We use the OLIA (Opportunistic Linked-Increases Algorithm) congestion control [26] which is a coupled one and is proven to be stable. For packet scheduling, we use the default *Lowest-RTT scheduler*.

A. Measurement Procedure

Figure 2 depicts our general procedure for the experiments with LKL on clients. While this procedure was originally designed for voice-activated traffic measurements and for MONROE project, it can be adapted to run on other measurement platforms and with other custom Linux network stacks.

At the beginning of the experiment, we run `tcpdump` to capture traffic on both client and server. The current LKL implementation uses the `virtio` device and communicates with the outside world through raw sockets. After bootstrapping the LKL, the main measurement application can run. It stores its results in log files. The next subsection elaborates this stage with our simulated voice-activated application (Fig. 3).

B. Measurements with Voice-activated Applications

Since the implementation of popular voice-recognition systems like Siri, Alexa, Google Assistant are closed-source and their traffic is encrypted, we use simulated traffic for our measurements. For this purpose, we modified the popular `iperf3` measurement software [27] on both clients and server [2]. The original `iperf3` software uses a separate control channel between the clients and the server. We modified this channel to exchange the experiment parameters at the beginning of each test as well as the results after the data transfer. Given that latency is an important factor in our measurements, we enabled `TCP NO_DELAY` socket option on the clients and the server. We also configured the Linux stacks with TCP Small Queue and Tail Loss Probe enabled.

The emulated traffic is based on our observations on Siri traffic pattern, as presented in II-A. As shown in Fig. 3, after a client connects to a server, it starts sending voice

[1]Merged code is available at https://github.com/hoang-tranviet/mptcp/commits/lkl_4.13-mptcp_v0.93_API

[2]Source code is available at https://github.com/hoang-tranviet/iperf-siri

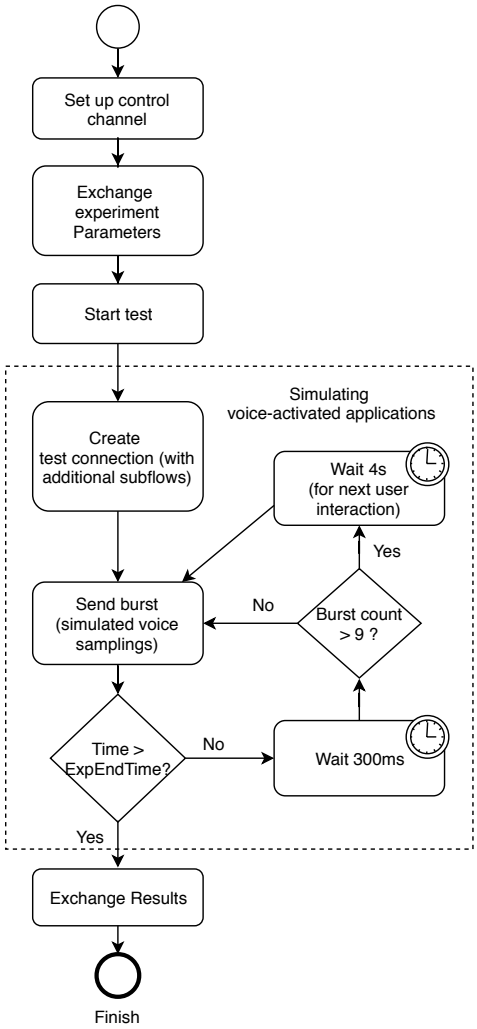
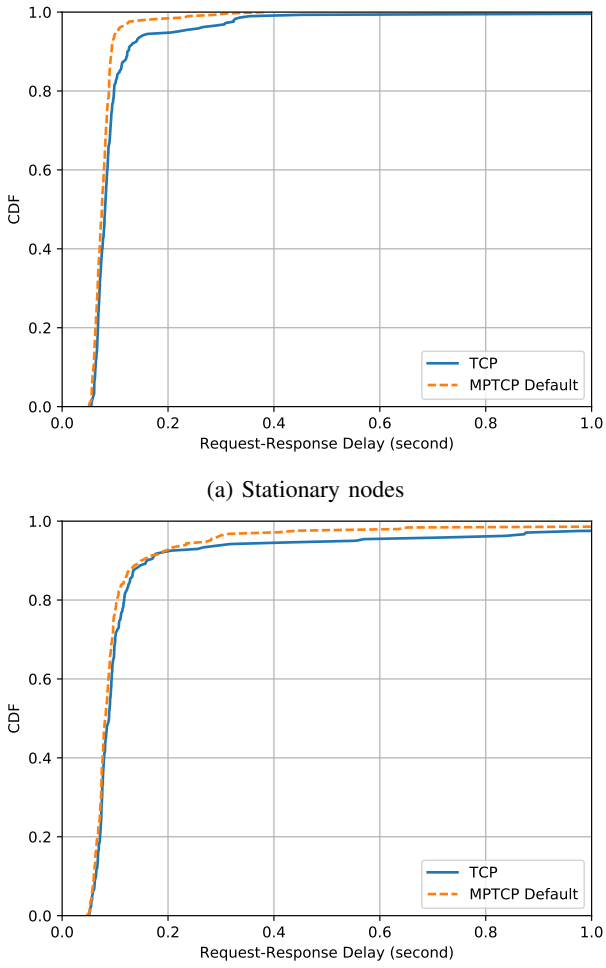


Fig. 3: Execution Flow of Measurement on client (simulating voice-activated applications)



(a) Stationary nodes

(b) Mobile nodes

Fig. 4: Request-Response Delay: MPTCP vs. TCP

data in a series of bursts. Each request consists of 9 bursts and each burst spans 10 TCP segments (ranging from 50 to 500 Bytes) on average. For every burst, the server may respond with a small reply corresponding to the HTTP `100 Continue` (Intermediate-Response). The inter-burst time is set to 300 msec. Once the server has received all request data, it immediately sends back a 750 Bytes response to the client. Our version of `iperf3` uses the enhanced socket API [25] to have more control on the creation of subflows. For users, the important metric is the request-response delay, which is defined as the delay between the transmission of the last burst and the arrival of the first response packet. A similar metric has been used in previous work [11], [12].

V. Sample Measurement Results

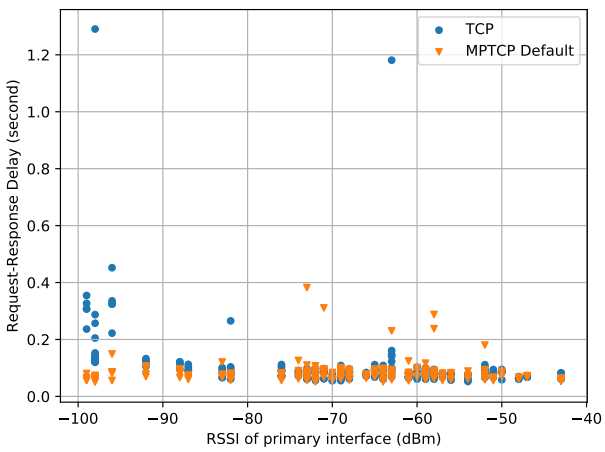
In this section we present some results of two measurement campaigns [3]. The first one compares the performance of TCP and MPTCP. The second campaign evaluates the performance of different configurations of MPTCP.

A. MPTCP versus TCP

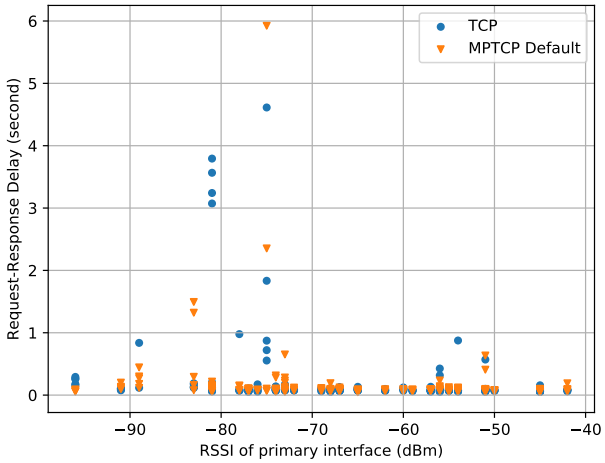
Initially, we ran the client application on 28 different *stationary* nodes towards our server. Each node performs five transactions with the server in 60 seconds (which gives five measured request-response delay samples), and the entire experiment is repeated three times. Figure 4a shows the CDF of the Request-Response Delays for this experiment.

Then, we ran the measurement with similar configuration but this time on 40 *mobile* nodes. As shown in Fig. 4b, it is clear that the average delay in this case is much higher than that of stationary nodes, with much longer tails that are not fully shown here. This is understandable given that the mobility of nodes likely reduces the connection quality, significantly increasing packet losses and delays. We can see that the default configuration of MPTCP delivers similar or better performance than TCP, though the difference is not always significant.

[3] All measurement scripts and corresponding collected data will be available at the publication time

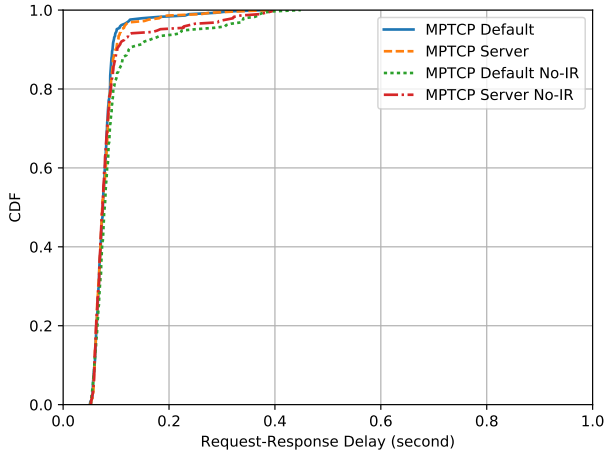


(a) Stationary nodes

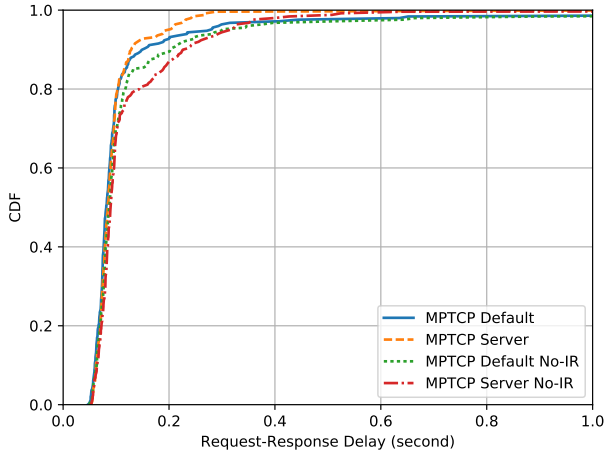


(b) Mobile nodes

Fig. 5: Request-Response Delay vs. Signal Strength



(a) Stationary nodes



(b) Mobile nodes

Fig. 6: Request-Response Delay: various MPTCP configurations

To dig in more details, we also collected the *Received signal strength indication (RSSI)* of the default cellular interface on each node, as shown in Fig. 5 and Table I. The high correlation between the delay and the signal strength in the case of TCP shows that the signal quality on the last-mile has a significant impact on the overall perceived delay. For MPTCP, there is no clear correlation between the user-level delay and the signal strength. This is likely the result of the scheduler decision, which tends to transfer data over the second subflow when the delay of the initial subflow increases.

Client	Stationary nodes	Mobile nodes
TCP	-0.34	-0.20
MPTCP Default	0.03	-0.11
MPTCP Server	0.07	-0.10
MPTCP Default No-IR	0.01	0.11
MPTCP Server No-IR	0.01	-0.10

TABLE I: Correlation between Request-Response Delay and the RSSI of default interface

B. Different MPTCP server configurations

In this campaign, we use two different schedulers on the server: *default scheduler* and *server scheduler*. The *default scheduler* prefers the subflow with the lowest round-trip-time. Meanwhile, the *server scheduler* [13] was designed for servers that serve mobile devices. This scheduler always prefers to transmit data over the last subflow on which it received data or a valid acknowledgement.

On the server side, the default (Lowest-RTT) scheduler may take undesired decisions in some cases. For example, consider a client that has both WiFi and cellular interfaces and initially sends data through the WiFi path. If the WiFi connectivity fails, e.g. because the user moves, then the server still sends traffic through this path since it does not know about the failed WiFi connection on the client side. The *server scheduler* [13] avoids this problem by choosing the subflow on which the server has just received client data. To be concrete, it remembers the timestamp of the latest original packet received on each subflow. A packet is considered original if it contains

new data based on its Data Sequence Number, or if it contains new Data-ACK that advances the left edge of sending window.

For any scheduler, the up-to-date information about each subflow should play an important role in improving data transfer performance. As described in Fig. 1, Siri server sends an *intermediate response* (`100 Continue`) after each received burst. To reveal the performance impact of these responses, we run the experiment with two different behaviors: servers send back these intermediate responses (default behavior) or do not send them back (the tests with "No-IR" annotation in the figures).

In all cases, there is no clear difference of delay in the first 70 percentiles. This represents the situations in which the default path is always the best path, so there is no impact of using different MPTCP configurations. The main differences are in the last 30% percentiles. In the case of mobile nodes (Fig. 6b), the *server scheduler* gives better performance for mobile nodes by keeping track of the most recent working subflow. However, for stationary nodes (Fig. 6a), the *server scheduler* gives nearly identical results since the connectivity status of each client interface is generally unchanged.

Additionally, when the server does not send *intermediate responses*, the delay increases significantly for both schedulers. Without the intermediate responses, the clients do not have up-to-date subflow information signaled from the server. In turn, the server has less information side to make a good decision when sending final response. The situation is worse for *server scheduler*, which just selects the most recently active subflow. While the *server scheduler* relies passively on the incoming traffic to select the subflow, the *intermediate responses* plays the role of the active probing on the current status of subflows.

VI. Conclusion

Voice-activated applications are on the rise and will likely play a more important role in the future. In this paper, we have proposed an extension of `iperf3` that models the network behaviour of such applications. We have then proposed an implemented a measurement methodology that leverages the LKL library to use a specific networking stack without changing the underlying Linux kernel. This is key to be able to test new protocols on platforms such as MONROE or Planetlab. We demonstrate the benefits of this approach on the MONROE platform. Our measurement results show that Multipath TCP with a proper configuration could help to improve the user-perceived delay in various network conditions.

Acknowledgments

This work has been partially funded by MONROE project (grant agreement No. 644399) supported by the European Commission, and *Action de Recherche Concertée* (ARC budget No. C1.31403.001-F) supported by Belgian Government.

References

[1] "Talk To Me: The Present & Future of In-Car Speech Recognition | Globalme." [Online]. Available: https://www.globalme.net/blog/the-present-and-future-of-in-car-speech-recognition

[2] "Agentbot - Automatic customer support with Artificial Intelligence." [Online]. Available: https://aivo.co/en/agentbot/

[3] J. P. Bigham *et al.*, "On How Deaf People Might Use Speech to Control Devices." ACM, 2017, pp. 383–384. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3132525.3134821

[4] A. Ford *et al.*, "TCP Extensions for Multipath Operation with Multiple Addresses," IETF RFC 6824, January 2013.

[5] Apple, "iOS: Multipath TCP Support in iOS 7," http://support.apple.com/en-us/HT201373.

[6] O. Bonaventure and S. Seo, "Multipath TCP deployments," *IETF Journal*, vol. 12, no. 2, pp. 24–27, 2016.

[7] O. Purdila, L. A. Grijincu, and N. R. I. C. R. . t. Tapus, "LKL: The Linux kernel library," in *Roedunet International Conference (RoEduNet), 2010 9th*, 2010, pp. 328–333.

[8] Y. Yamada *et al.*, "Development and Evaluation of Julius-Compatible Interface for Kaldi ASR," in *International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. Springer, 2017, pp. 91–96.

[9] A. Langley *et al.*, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 183–196.

[10] L. Caviglione, "A first look at traffic patterns of Siri," *Transactions on Emerging Telecommunications Technologies*, vol. 26, no. 4, pp. 664–669, 2013. [Online]. Available: http://doi.wiley.com/10.1002/ett.2697

[11] M. Assefi, M. Wittie, and A. Knight, "Impact of network performance on cloud speech recognition," in *Computer Communication and Networks (ICCCN), 2015*. IEEE, 2015, pp. 1–6.

[12] M. Assefi *et al.*, "An experimental evaluation of Apple Siri and Google speech recognition," *Proccedings of the 2015 ISCA SEDE*, 2015.

[13] Q. De Coninck and O. Bonaventure, "Tuning multipath TCP for interactive applications on smartphones," in *IFIP Networking 2018*, http://hdl.handle.net/2078.1/197489.

[14] C. Paasch *et al.*, "Exploring Mobile/WiFi Handover with Multipath TCP," in *ACM SIGCOMM workshop CellNet*, 2012, pp. 31–36.

[15] C. Raiciu *et al.*, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP," in *USENIX NSDI*, 2012.

[16] B. Hesmans *et al.*, "Smapp: Towards smart multipath tcp-enabled applications," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. ACM, 2015, p. 28.

[17] C. Paasch *et al.*, "Experimental evaluation of Multipath TCP schedulers," in *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, ser. CSWS '14. New York, NY, USA: ACM, 2014, pp. 27–32.

[18] C. Paasch, S. Barre *et al.*, "Multipath TCP implementation in the Linux kernel," 2014, available from http://www.multipath-tcp.org.

[19] Ö. Alay *et al.*, "Measuring and assessing mobile broadband networks with MONROE," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2016 IEEE 17th International Symposium on A*. IEEE, 2016, pp. 1–3.

[20] C. Boettiger, "An introduction to Docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015.

[21] M. Peón-Quirós *et al.*, "Results from Running an Experiment as a Service Platform for Mobile Networks," in *Proceedings of the 11th Workshop on Wireless Network Testbeds, Experimental evaluation and Characterization*. ACM, 2017, pp. 9–16. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3131473.3131485

[22] A. S. Khatouni *et al.*, "Speedtest-like Measurements in 3G/4G Networks: the MONROE Experience," in *Teletraffic Congress (ITC 29), 2017 29th International*, vol. 1. IEEE, 2017, pp. 169–177.

[23] R. Davoli, "VDE: Virtual Distributed Ethernet," in *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005*. IEEE, 2005, pp. 213–220.

[24] D. Intel, "Data plane Development Kit," 2014. [Online]. Available: https://dpdk.org/

[25] B. Hesmans and O. Bonaventure, "An enhanced socket API for Multipath TCP," in *ANRW*. ACM, Jun. 2016, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2959424.2959433

[26] R. Khalili *et al.*, "MPTCP is not pareto-optimal: performance issues and a possible solution," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 2012, pp. 1–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2413178

[27] ESnet. iperf3. [Online]. Available: http://software.es.net/iperf/