Tuning Multipath TCP for Interactive Applications on Smartphones

Quentin De Coninck*, Olivier Bonaventure
Institute of Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM)
Université catholique de Louvain
Louvain-la-Neuve, Belgium
Email: {quentin.deconinck,olivier.bonaventure}@uclouvain.be

***Abstract*—Multipath TCP enables smartphones to simultaneously use both WiFi and LTE to exchange data over a single connection. This provides bandwidth aggregation and more importantly reduces the handover delay when switching from one network to another. This is very important for delay sensitive applications such as the growing voice activated apps. On smartphones, user experience is always a compromise between network performance and energy consumption. However, the Multipath TCP implementation in the Linux kernel was mainly tuned for bandwidth aggregation and often wakes up the cellular interface by creating a path without sending data on it.**

In this paper, we propose, implement and evaluate MultiMob, a solution providing fast handover with low cellular usage for interactive applications. MultiMob relies on three principles. First, it delays the utilization of the LTE network. Second, it allows the mobile to inform the server of its currently preferred wireless network. Third, MultiMob extends the Multipath TCP handshake to enable immediate retransmissions to speedup handover. We implement MultiMob on Android 6 smartphones and evaluate its benefits by using both microbenchmarks and in the field measurements. Our results show that MultiMob provides similar latency as the standard Linux implementation while significantly lowering the cellular usage.

I. Introduction

Mobile devices such as smartphones are now an integral part of our digital life. Mobile data traffic continues to grow [1]. The performance of the WiFi and cellular networks have significantly increased over the last years. Compared with 3G, LTE provides both higher bandwidth and lower latency while WiFi reaches Gbps and more. These high bandwidth and low-latency networks encouraged the deployment of new applications. Mobile video benefited a lot from the bandwidth improvements. On the other hand, the lower latency enabled a new family of voice activated applications [21]. The user uses his/her voice instead of buttons to interact with the application that sends voice samples to the cloud. For such applications, latency is key and many protocols have been tuned during the last years to reduce it [4], [33].

For most smartphone users, the WiFi and cellular networks are not equivalent. WiFi has two major advantages compared to cellular networks. First, using WiFi consumes less energy [20], [26]. Second, most service providers charge for cellular data while most WiFi networks are free or charged on a flat-rate basis. For these reasons, many smartphones owners only use their cellular interface for voice calls and when there is no WiFi network available [7].

Multipath TCP is a recent TCP extension [16] that was designed with these smartphones in mind. Pluntke et al. [32] and then Raiciu et al. [35] first discussed the expected benefits of Multipath TCP on such mobiles devices. Later, Paasch et al. implemented handover features [30] in the Linux kernel [29]. Lim et al. showed the importance of taking energy consumption into account [23].

Industry has already adopted Multipath TCP on smartphones with two major deployments [3]. Apple uses Multipath TCP for its Siri voice activated application since 2013 and enables Multipath TCP for any application on iOS11 [7]. This is the largest deployment of Multipath TCP today [3] with about 700 million devices. There is another major deployment in Korea. In this country, high-end Android smartphones use Multipath TCP through network-operated SOCKS proxies to achieve Gbps [37].

Many authors studied and tuned the bandwidth aggregation capabilities of Multipath TCP [36], [22], [18], [27], [11], [6] in mobile networks. Although popular in the scientific literature, this is not the main use case for Multipath TCP on mobiles [7]. Smartphones rarely exchange large files [9] that would benefit from bandwidth aggregation. Measurement studies [13], [9] show that smartphones mainly use either short or long-lived intermittent TCP connections. Apple recently opened Multipath TCP on iOS11 mainly to provide seamless handovers and support interactive applications [7].

We first describe the current state-of-the-art of Multipath TCP on smartphones in Section II. We then propose MultiMob, a series of improvements that adapt Multipath TCP to the requirements of today's smartphone applications. More precisely, MultiMob provides a good compromise between latency and cellular usage.

A MultiMob server replies on the subflow used by the smartphone (§ III-A). If a smartphone sends a request over a cellular subflow because its WiFi subflow performs badly, the server should send its reply over the same subflow.

MultiMob minimizes cellular usage and unused subflows (§ III-B). Like iOS11 [7], MultiMob prefers to use the WiFi interface over the cellular one. MultiMob replaces the *make-before-break* strategy of the Multipath TCP implementation on Linux by *break-before-make*. With this strategy, the cellular

* FNRS Research Fellow

interface is only used after a failure of the WiFi one.

MultiMob limits handover delays (§ III-C). The *break-before-make* strategy minimizes energy consumption but at the expense of increased handover delays. MultiMob reduces those delays by extending the Multipath TCP protocol to carry data during the subflow handshake.

In Sect. IV, we collect measurements in a Mininet environment to assess MultiMob characteristics. In Sect. V, we evaluate MultiMob with real smartphones. Finally, Sect. VI concludes this paper. An extended technical report of this work is available [10].

II. State of the Art and Motivation

Multipath TCP [16] was designed with multihomed devices such as smartphones in mind. It enables them to exchange data belonging to a single connection over different network paths. It is described in details in [16], [36]. We briefly summarize its main features here. A Multipath TCP connection is in fact a combination of different TCP connections, called *subflows* in [16], that are grouped together. A Multipath TCP connection is established by using a three-way handshake as a regular TCP connection, except that the SYN packet contains the `MP_CAPABLE` option. This option negotiates the utilization of Multipath TCP and allows the client and server to exchange keys. Each Multipath TCP connection is identified by a token that is derived from the keys exchanged during the initial handshake [16]. Data can be exchanged over the initial subflow and both the client and the server can create additional subflows to use other paths or perform handovers. Those additional subflows must be established by using a four-way handshake with SYN packets that contain the `MP_JOIN` option. This option includes a token that identifies the corresponding Multipath TCP connection. Data can be transmitted over any of the available subflows. Multipath TCP uses two levels of sequence numbers. The standard TCP sequence and acknowledgement numbers are used in the TCP header to handle data sequencing and retransmissions on a per-subflow basis. Furthermore, Multipath TCP uses the Data Sequence Number (DSN) that tracks the position of the data in the bytestream. The DSN is placed inside the Data Sequence Signal TCP Option that also carries DSN acknowledgements. Thanks to this DSN, Multipath TCP can transmit data over one subflow and later retransmit it over another subflow because the initial one failed or became unresponsive. *Reinjecting* data over a different subflow is key to support handovers [36], [30].

There are two main implementations of Multipath TCP on mobile nodes: Apple's implementation on iOS [7] and the open-source Linux implementation [29]. We focus on the latter because it fully implements the protocol and can be easily modified. Besides supporting all the features described in [16], it includes several heuristics that are important for performance [36] without impacting interoperability.

A first component of the Multipath TCP implementation in the Linux kernel is the *path manager*. It determines when additional subflows must be created. The initial subflow is always established on the interface that points to the current default route. On a client, the default `fullmesh` path manager [29] creates new subflows immediately after the creation of the initial one and each time a new IP address is assigned to the client or learned from the server. This path manager does not initiate subflows from the server because it expects that the client's firewall will block incoming TCP connections.

A second component is the *packet scheduler*. It decides on which established subflow the next packet will be sent. The `default` scheduler extracts the smoothed round-trip-time of all the subflows whose congestion window is not full and selects the one having the lowest smoothed round-trip-time (RTT). Other schedulers more adapted to heterogeneous paths have been proposed [25], [28], [15].

Multipath TCP [16] also supports *backup subflows*. When a subflow is established, it is possible to set a bit in the `MP_JOIN` option to indicate that this subflow should not be selected by the scheduler to exchange data unless all non-backup subflows have failed. We observed that Siri in iOS11 [7] also sets the backup bit on the cellular subflow to discourage the utilization of the cellular interface to transport data. In the Linux Multipath TCP implementation, a subflow is considered to be in a *potentially failed* state once its retransmission timer expires. This subflow transitions to the active state as soon as new data is acknowledged on the subflow. The default scheduler uses a backup subflow if all the regular subflows are in the *potentially failed* state.

A. Multipath TCP on Smartphones

Before tuning Multipath TCP on smartphones, it is important to understand how they interact with the network. We summarize in this section some of the lessons we learned based on discussions with network operators and previous works.

Smartphone applications rarely perform bulk transfers Multipath TCP was designed to aggregate bandwidth and many articles evaluated whether Multipath TCP reaches that objective [31], [6], [11], [34]. However, smartphones rarely exchange very long files [14], [9]. Most of the connections carry a few KB. Many connections also experience large idle times [8]. While not being an issue from TCP perspective, from an energy viewpoint, it can consume energy if the radio needs to remain active to support it.

Many subflows do not carry data The `fullmesh` path manager immediately creates subflows on all active interfaces. However, most of these subflows are useless, i.e., *no* data is sent over them [9]. With the `default` scheduler, if the initial subflow exhibits a lower RTT than the additional ones, Multipath TCP will only use the initial one. Previous works also indicate than Multipath TCP can perform worse than TCP on short flows in heterogeneous networks [18], [27].

Mismatch with user expectations Most users favor WiFi over cellular for both monetary and power consumption reasons [20], [7]. They expect that their smartphone will use WiFi whenever it works well and will switch to cellular only if it brings some benefits. However, the packet scheduling decision is taken by the sender of the packet. In practice, smartphones mainly receive data [14], [9], meaning that most of the

scheduling decisions are taken by remote servers. Because the measured round-trip-time is the only metric, the server scheduling decision can go against the user expectations.

Backup subflows consume energy One way to minimize the utilization of the cellular network is to always establish the cellular subflow as a backup subflow [16], [23]. While useful in mobility scenarios, there is no point to create backup subflows if the primary one does not face any connectivity issue. Indeed, energy consumption is a major concern for mobile devices [7], [5], [2]. However, opening a subflow on the cellular interface without using it is expensive from an energy consumption viewpoint [11], the WiFi interface consuming at least five times less than the LTE one [26]. In the remaining of this paper, we use the LTE model proposed by Huang et al. [20] to estimate the cellular power consumption (we expect similar results with other models [26]). In the model used [20], opening one cellular subflow on a smartphone is equivalent to lighting up the screen 100% during the `RRC_CONNECTED` period, which lasts around 11 s. Opening preventively the cellular subflow as proposed in [30] is thus very expensive from an energy consumption viewpoint. Siri in iOS11 still creates cellular subflows at the beginning of the connection.

Related Works Lim et al. [24] proposed eMPTCP that delays the use of the cellular below a given threshold of bytes transfered and opens the cellular subflow if the WiFi bandwidth is not sufficient. While working with bulk transfers, interactive applications can transmit very few bytes and do not need large bandwidths. Sinky et al. [38] proposes to rely on the signal strength of the WiFi network to tune the congestion window to trigger seamless WiFi handover with bulk transfer. However, it was only tested under NS3-DCE environment and not with actual devices. Han et al. [17] proposes to disable the cellular when the WiFi is sufficient with delay-tolerant traffic. However, interactive traffic is delay-sensitive.

III. Tuning Multipath TCP

We explain how MultiMob improves Linux Multipath TCP. We first add to the server's packet scheduler a heuristic that enables it to infer the wireless conditions that affect the client subflows. Second, we implement an oracle that monitors the network and opens cellular subflows only when needed. Third, we extend the Multipath TCP protocol so that a client can retransmit data inside the SYN that is used to create an additional subflow during a handover.

A. Towards Global Scheduling

When a Multipath TCP connection is composed of 2 or more subflows, each of the communicating hosts independently selects the best subflow to transmit each data. The Linux implementation selects the available subflow with the lowest round-trip-time (RTT). This scheduler works well in a variety of environments [31]. However, selecting subflows only on the basis of their RTT is not always the best solution. Consider a smartphone user that moves while using the Siri application. This application regularly sends small bursts of data and the server returns responses. If the smartphone detects that the WiFi starts to be lossy, it will start to send data over the cellular subflow. However, the server is not aware of the movement of the smartphone and its packet scheduler still sends responses over the WiFi subflow because it has the lowest RTT. The server will only switch to the cellular subflow after the expiration of its retransmission timer, which potentially wastes hundreds of milliseconds.

To solve this problem, MultiMob includes a packet scheduler that uses the most recent data sent by the smartphone as a hint to select the most suitable subflow. On the smartphone, MultiMob uses a priority scheduler that favors WiFi and only uses cellular when the WiFi subflow experiences retransmissions. The server-side scheduler maintains for each subflow the timestamp of the *last original packet* received over this subflow. A packet is considered to be *original* if it contains new data (based on its DSN) or if it successfully concludes a subflow establishment. Similarly, an acknowledgement is considered to be original if its Data ACK advances the lower edge of the sending window. The `MultiMob` scheduler first removes from consideration the potentially failed subflows and the ones where this data has already been transmitted. Then it iterates over all remaining subflows to identify the one having the most recent original reception. If the congestion window of this subflow is not full, it is selected.

Thanks to this scheduler, the server can quickly detect the most suitable subflow while taking into account subflow backup preferences. For an interactive application like Siri that sends small requests, the server will always reply on the subflow that was last used by the client.

B. Break-Before-Make

In the Linux kernel implementation, when a Multipath TCP connection starts, the `fullmesh` path manager opens the connection over the primary interface and then creates subflows over the other ones. If the cellular interface is configured as a backup interface, data packets will only be sent over this interface once the WiFi interface fails. This *make-before-break* approach minimizes the amount of data sent over the cellular interface. Unfortunately, it does not minimize the energy consumption. There is no significant difference from an energy consumption viewpoint between a cellular interface that transmits only SYN/FIN or several data packets.

MultiMob opts for *break-before-make* and creates subflows over the backup interface after having detected failures on the primary interface. With *break-before-make*, the key issue from a performance viewpoint becomes how quickly can the smartphone detect that a wireless interface works badly and new backup subflows must be created. MultiMob detects those failures through a *Multipath TCP oracle* implemented as a kernel module. The oracle relies on the assumption that if a network interface experiences connectivity issues, subflows using it will experience retransmissions and losses, even if they belong to different connections. To track those events, our oracle maintains a monitoring table of netpaths. A *netpath* is a tuple (IP_{src}, IP_{dst}, network interface). We aggregate the information on a per layer-3 flow basis to reduce the size of the

monitoring table. This structure is well adapted to deployments with SOCKS proxies such as [37] where all Multipath TCP connections are terminated on the proxy.

Our oracle computes every T_s seconds statistics based on the subflows associated to a given netpath. Our current implementation collects three metrics: smoothed loss rate ($sloss$), smoothed retransmissions rate ($sretrans$) and maximum RTO. Those statistics are computed based on the per-subflow state maintained by the kernel. It also takes into account Tail Loss Probes [12]. When the TLP timer fires, we enter FACK mode and the packet at the head of the write queue is marked as lost. The smoothed rates are computed by using Volume-weighted Exponential Moving Averages (V-EMA) used by Android to estimate the loss rate of WiFi beacons. These V-EMA reduce to the three following equations

$$val_{i+1} = \frac{prod_{i+1}}{vol_{i+1}} \quad (1)$$

$$prod_{i+1} = \alpha(val_{new} \cdot vol_{new}) + (1-\alpha)prod_i \quad (2)$$

$$vol_{i+1} = \alpha \cdot vol_{new} + (1-\alpha)vol_i \quad (3)$$

where val_{new} is the new value of the studied metric (e.g., lost sent packets during the last T_s period), vol_{new} is the volume of this new value (e.g., total number of packets sent during last T_s period), $prod_i$ is the product at iteration i, vol_i the volume at iteration i and val_i the value at iteration i. $prod_0 = vol_0 = val_0 = 0$ and no value is computed if vol_{new} is 0. $\alpha \in [0, 1]$ is a parameter experimentally set to 0.5 as in Android.

The MultiMob oracle sets thresholds to detect underperforming netpaths. Once a threshold is crossed, MultiMob triggers the creation of backup subflows for all connections associated to the underperforming netpath. It also marks the subflows associated with that netpath as potentially failed. Since the oracle is part of the kernel, it can query the state of all established Multipath TCP connections and trigger backup subflows creation once a problem is detected.

The last scenario that we consider is a client-initiated download. During such a download, the server pushes data towards the client. If a subflow fails, the client stops receiving data, but it is difficult for the Multipath TCP stack to distinguish between losses in the network and the server application becoming idle for any reason. We modify Multipath TCP so that the server can indicate to the client that a data transfer is not yet finished. This can work with existing applications. We define two signals. The first one is sent in the Multipath TCP DSN option. We modify one of the unused bits of the DSN option that we call the `MP_IDLE` bit. This bit is set by server when it sends a data packet that empties its send buffer. Otherwise, the `MP_IDLE` bit of the DSN option is reset. Since this bit is included in the DSS option, it is sent reliably to the client. A receiver should not expect a connection to be idle unless it has received a DSN option with the `MP_IDLE` bit set. We also define a new experimental Multipath TCP option that carries the current value of the RTO. The client sends an empty RTO option from time to time and the server returns the same option containing its current retransmission timer. The client uses this information to set its idle timer at $max(500 \text{ msec}, RTO_{Server})$. This timer runs while the `MP_IDLE` flag of the last received data is reset. It is reset every time a packet is received on the subflow and stopped if the last data packet had the `MP_IDLE` flag set. If the timer expires, the oracle triggers the creation of backup subflows.

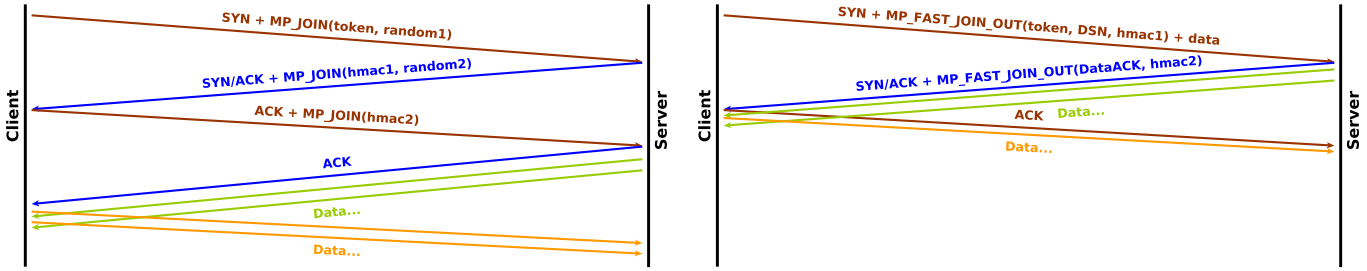
C. Immediate reinjections

The *break-before-make* approach described in the previous section is beneficial from an energy viewpoint. However, a mobile typically detects the failure of a wireless interface by the expiration of its retransmission timer or TLP probe. This unacknowledged data can only be retransmitted over another interface once a subflow has been established over this interface. Multipath TCP [16] requires a four-way handshake before allowing the transmission of data from the server. This handshake has two purposes. First, it creates state on the endpoints (and possibly on the intermediate middleboxes). Second, the client and the server authenticate each other. This authentication is performed by using the keys exchanged during the initial handshake. Both the client and the server exchange HMACs computed over these keys and random numbers exchanged in the SYN and SYN+ACK (see Fig. 1a). Unfortunately, this handshake delays the reinjection of the lost data since the client cannot send data before having received the fourth ACK [16].

To reduce this delay, we modify Multipath TCP to support the transmission of data inside SYN or SYN+ACK packets. For this, we define two new Multipath TCP options: `FAST_JOIN_IN` (`FJI`) and `FAST_JOIN_OUT` (`FJO`). This is different than TCP Fast Open [33] because a new subflow is established between hosts that already share state for one Multipath TCP connection.

A naive approach would be to simply place data inside the SYN and require the server to accept this data immediately. Unfortunately, this solution would cause security problems because this SYN is not authenticated. The `MP_JOIN` option that it carries contains only the 32 bits token that identifies the connection and a random number that is used to authenticate the server (Fig. 1a). This token is not sufficient to authenticate the client because a passive listener could have observed it during the establishment of a previous subflow for the same connection, e.g., on an open WiFi network.

The `FJO` option described on Fig. 1b solves this problem and allows the client to carry authenticated data in the initial SYN. Our `FJO` option contains three fields. The token identifies the Multipath TCP connection as in the `MP_JOIN` option. The Data Sequence Number (DSN) indicates the sequence number of the data contained in the SYN payload. The third field is a HMAC computed over the connection keys exchanged during the initial handshake and the DSN. This last field ensures that the initiator of the subflow is one of the connection hosts. To prevent replay attacks, our implementation only accepts one SYN containing the `FJO` option for a given DSN. To cope with lost acknowledgments, if $EDSN$ is the next expected DSN on the server and $SDSN$



(a) Multipath TCP uses a four-way handshake that lasts two round-trip-times to create an additional subflows with `JOIN`.

(b) With `FAST_JOIN`, the client can immediately send data inside the SYN packet.

Fig. 1: Time-sequence diagrams for the establishment of additional subflows.

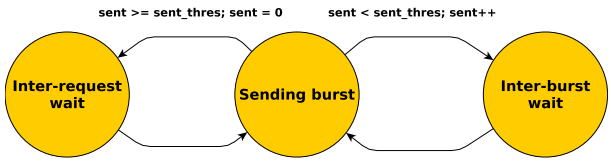


Fig. 2: State machine of a simple interactive application.

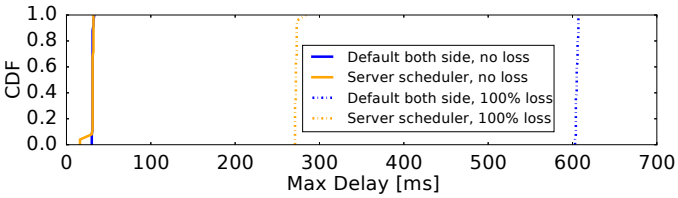


Fig. 3: Maximum delay to return an answer to simplified interactive requests. Each scenario ran 25 times. Losses begin to occur when the client is in *inter-user interaction wait* state, i.e., between request bursts.

the DSN contained in the `FAST_JOIN` SYN, the server allows $SDSN$ to be in the range $[EDSN - rcv\_wnd, EDSN]$ where rcv_wnd is its receive window. Once the SYN has been acknowledged, the server can immediately start to send data to the client.

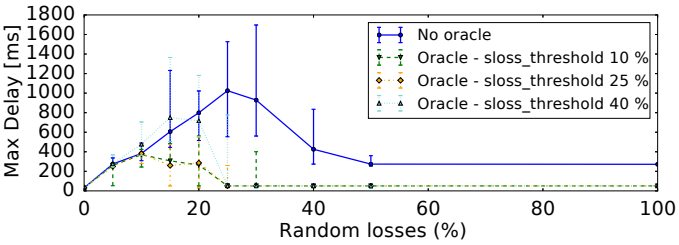
The `FJO` option is useful when the mobile client sends data to a server. However, there are situations where the server pushes data towards the client. A typical example are streaming applications where the server pushes data at a regular rate. When the oracle running on the mobile client detects losses or the absence of data, it may want to quickly establish a subflow without having data to send to the server. This case is covered with the `FJI` option [10] (not shown for space limitations). This option is very similar to the `FJO` option, except that it contains the current Data ACK instead of a DSN, with the HMAC computed over this Data ACK. With this new option, the server can authenticate the client immediately and send data upon reception of the SYN. By using the `FJI` option, the data transfer can resume after 1 RTT, instead of 2 RTTs with normal join.

IV. Emulations

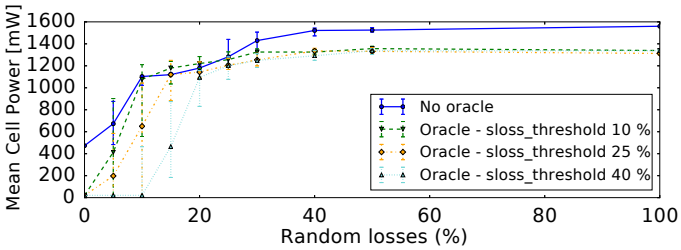
We evaluate in this section the performance of MultiMob in Mininet environments [19] in a scenario with two disjoint paths between the client and the server. Emulations are based on Multipath TCP v0.91 in Linux 4.1.

a) Studied Traffic: Siri is a famous example of interactive traffic. However, it is not open-source and only runs on iOS devices. To evaluate the interactions between a simple interactive application and Multipath TCP, we use a simplified model based on an analysis of the behavior of Siri. Our model is a three-states process shown in Fig. 2. The client maintains a counter: `sent`. In the *sending burst* state, the client sends a burst of 2500 bytes using packets carrying between 50 and 500 bytes. Then `sent` is incremented and the client waits in the *inter-burst wait* state before going back to *sending burst*. Once `sent` reaches `sent_thresh`, the client switches to the *inter-user interaction wait* state that represents the random delay between successive user interactions. `sent_thres`, *inter-burst wait* and *inter-user interaction wait* are empirically set to 9, 1/3 s and 5 s respectively. We model the server as a process that returns a 750 bytes response after each burst. Our simple client application then collects the delay between each request and the server's response. Unless stated, all the measurements in this paper are based on this traffic.

b) MultiMob Scheduler: The primary path exhibits a RTT of 15 ms and the additional one 25 ms. Both paths have a bandwidth of 10 Mbps and the router queue sizes are equal to the bandwidth-delay product. The client opens the connection over the primary path and then creates a backup subflow over the additional one. Figure 3 shows that when there are no losses, the `default` and `MultiMob` schedulers running on the server exhibit quite similar performances. Notice that because of the default `tcp_slow_start_after_idle` set to 1, a request can be answered in two RTTs if its sending phase generates more packets than the initial congestion window (10 packets). However, when the primary subflow fails between two requests, the `MultiMob` scheduler reduces the maximal delay experienced by a factor of two. When the client sends its first request after a loss, it experiences a RTO before reinjecting the packet on the additional subflow. However, with the `default` scheduler, the server does not



(a) Maximal delay to answer a simplified interactive request.



(b) Estimated mean cell power consumption based on model [20].

Fig. 4: Simplified interactive requests with light continuous background traffic. The second interface is set as backup. If any, the loss event occurs while the client is in *inter-user interaction wait* state, i.e., between request bursts. Markers shows medians and error bars 25th and 75th percentiles over 25 runs.

know that the primary subflow failed, and it sends the reply to the primary lossy subflow and experiences a RTO too before reinjecting on the additional subflow. Since the `MultiMob` server-side scheduler follows the last client decision, it does not experience the RTO at the server side.

c) Influence of Threshold Value: To assess the benefits of the oracle and determine the threshold value for $sloss$, we rely on simplified interactive requests while a light background request/response traffic (12 KB/s) is present. Figure 4a shows the maximal latency to answer a request and Fig. 4b shows the estimated mean power consumed on the second path. The energy consumption is estimated by using the packet trace and the model presented in [20], considering that the cellular interface is always powered on. Without losses, we observe similar requests delays, while the backup subflow is not established with the oracle. When losses occur on the primary path, the oracle knows that the background traffic experiences connectivity issues and creates backup subflows for all connections using that path. Then, the simulated interactive client can directly use the additional path and does not face RTO. Since the server uses `MultiMob` scheduler, it replies on the subflow used for the request and no RTO occurs. On the contrary, the interactive connection must face a RTO if there is no oracle before using the additional path, even if the additional subflow is always established at the beginning of the connection. Furthermore, when the link is very flappy (20-30% losses), the case without the oracle tries to reuse the lossy path once some ACKs manage to reach the host, while the oracle prevents this behavior. With the oracle the creation of the additional subflow depends on the network conditions and the $sloss$ threshold. When set to a low value, e.g., 10%, delays remain low but a few losses suffice to open the additional subflow. With higher values like 40%, the additional path remains closed in the median case when the primary path experiences 10% of random losses, but it can experience higher latencies. Based on those simulations, we experimentally set the $sloss$ threshold to 25% as a reasonable trade-off between low-latency and low additional path use. $sretrans$ is set to 50% and the max RTO threshold is empirically set to 1.5 seconds to avoid using a subflow that might hurt interactivity because of lack of retransmission reactiveness.

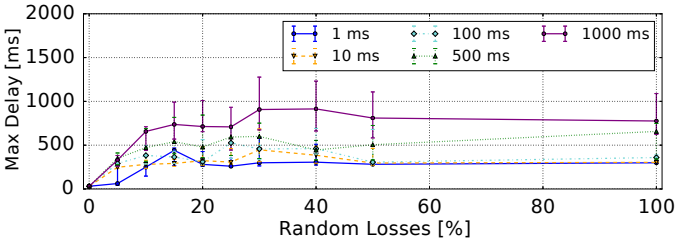


Fig. 5: Varying T_s for interactive traffic, with $sloss$ set to 25%. Showing 25th, 50th and 75th percentiles.

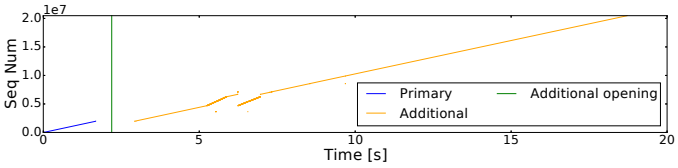


Fig. 6: Time-sequence graph of the packets received by a client during a 20 MB HTTP GET. The primary subflow suffers from 100% losses at 1.5 s. Retransmissions at 6 second are caused by a burst of duplicate ACKs.

d) Influence of the Oracle Periodicity: The reactivity of the oracle also depends on the oracle timer T_s. Indeed, as shown in Fig. 5, the lower T_s, the quicker the reaction of the oracle to losses and the lower the variability of the detection. A value of 1 ms allows very quick reaction, but the oracle might spend a lot of CPU time to update its monitoring table. In the remaining of the paper, T_s is empirically set to 500 ms to match sub-second reactivity and low CPU usage on mobiles.

e) Bulk Download and Primary Subflow Loss: The client downloads a 20 MB file and we add 100% losses on the primary path after 1.5 s. Fig. 6 shows that after some idle time, the client detects that it did not receive data and triggers the creation of a second subflow. The server then starts to use the new subflow and the data transfer continues.

f) Fast Join Benefits: We evaluate the benefits of using the `FAST_JOIN_OUT` option with regular request/response traffic over an emulated network where the primary path experiences 100% losses after five seconds. Figure 7 shows the difference of the delays of the first request following the

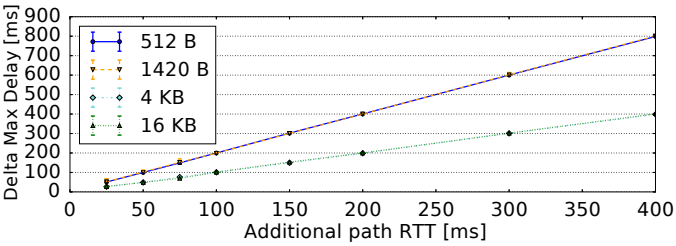


Fig. 7: Delta of max delay between normal and fast joins depending on the request size. Markers are medians over 6×2 runs, bars show min and max.

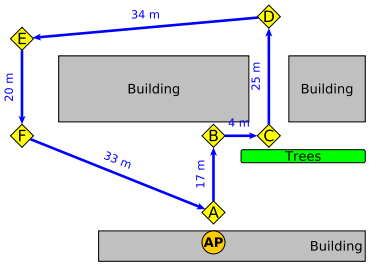


Fig. 8: Walk map for micro-benchmarks.

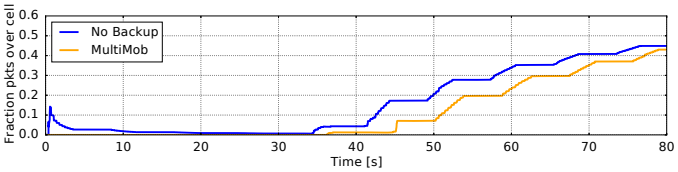
loss event using normal and fast joins. If the request fits inside one TCP packet, as for the popular Siri application, the fast joins provide immediate reinjections when the client sends data and the response can be received after one RTT.

V. Performance Evaluation

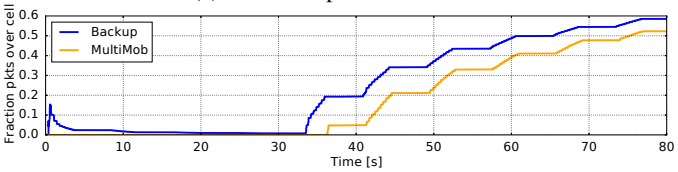
This section presents the evaluation of MultiMob on Nexus 5 smartphones running Android 6.0.1. For this, we backported Multipath TCP v0.89 to the Linux 3.4 kernel of Nexus 5 phones. Three configurations are studied: 1) *No backup* (*NBK*), MPTCP with the `fullmesh` path manager; 2) *Backup* (*BK*), NBK with backup subflows on cellular; and 3) *MultiMob*, the proposed solution described in Sect. III. We use two servers. The first one, configured with the `default` scheduler, is used by NBK and BK. The second one, configured with the `MultiMob` server-side scheduler, is used by MultiMob. In this section, we first explore particular use cases with micro-benchmarks to understand the benefits of MultiMob. We then compare at a larger scale NBK and BK with MultiMob through active measurements performed on a set of modified Android 6 smartphones used by real users.

A. Mobility Micro-Benchmarks

To evaluate how MultiMob performs in changing wireless conditions, we go for a short walk (Figure 8) with two smartphones. The first uses MultiMob and the other a vanilla Multipath TCP configuration. Our walk starts at **A**, close to the WiFi AP. Starting from **C**, the WiFi signal becomes weaker given the distance and the presence of trees and buildings. Android usually detects the loss of the WiFi signal and tears down the WiFi network at location **D**. Starting at location **F**, the WiFi signal becomes available again.



(a) No Backup vs. MultiMob.



(b) Backup vs. MultiMob.

Fig. 9: Evolution of the mean fraction of total packets carried by the cellular network for the simplified interactive traffic.

Configuration	MD (ms)	RA	CP (mW)
No backup	1112	100	884
Backup	780	100	885
MultiMob	1183	100	657

TABLE I: Aggregated results from simulated interactive micro-benchmarks. MultiMob shows the mean value over both runs. MD = Max Delay, RA = Requests Answered, CP = mean Cell Power consumption.

Simulated Interactive Traffic Our test phones send 100 requests during our 80 s walk from **A** to **D**. Figure 9 shows the instantaneous mean over the test duration of the fraction of total packets that are carried by the cellular interface for the two runs. In addition, Tab. I shows aggregated results related to these tests. With NBK and BK, the cellular subflow is always created at the beginning of the connection, but no data packet is sent on the cellular subflow while the WiFi signal remains good. This is expected for the backup case, and the larger RTT on the cellular network combined to the low network load explain the NBK results. When the client requests start to be lost between locations **C** and **D**, the cellular network is used to recover the connectivity. Since the cellular subflow was established at the beginning of the connection, the NBK and BK cases often experience a lower maximal delay than MultiMob. Since the cellular subflow is already established, the NBK and BK cases can reinject requests on the cellular subflow as soon as a RTO occurs on the WiFi subflow. MultiMob needs first to detect the connectivity loss with its oracle before establishing the cellular subflow, but its maximum delay remains similar to those of NBK and BK cases[1]. On the opposite, MultiMob consumes less cellular energy since it delays the utilization of the cellular interface.

Fixed Rate Streaming Traffic For this test, we configure the smartphones to stream a web radio over HTTP while performing twice the walk presented in Fig. 8. Our servers relay the same web radio at a fixed bitrate using Icecast.

[1]It is actually very dependent of the wireless conditions of a particular run. The lowest max delay observed for MultiMob over runs was 599 ms.

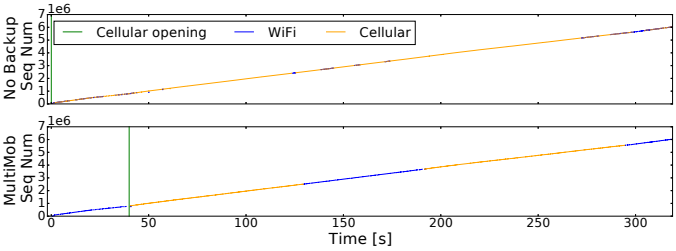


Fig. 10: Time-sequence graph of the server streaming flow as perceived by the client for the No Backup vs. MultiMob. Color indicates on which interface packet was received.

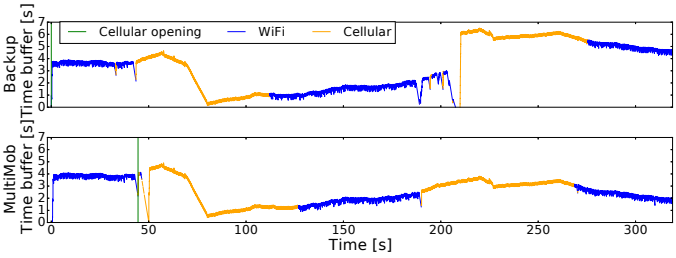


Fig. 11: Playing time of the client buffer for the worst case in Backup vs. MultiMob test. Color indicates on which interface packet was last received.

Since all the data flows from the server to the client, all the scheduling decisions are made by the server.

Figure 10 shows the time-sequence graph for the NBK vs. MultiMob test. We observed no stall during those experiments. However, the NBK case sends data nearly exclusively on the cellular interface, even when the WiFi network is available. From the server perspective, the cellular network appears to be more stable with an often lower estimated RTT than the WiFi one due to motion. This explains why the `default` scheduler prefers the cellular subflow. MultiMob forces the server to use the WiFi when it is still available. The WiFi to cellular (between **C** and **D**) and the cellular to WiFi (between **F** and **A**) handovers are visible on the MultiMob graph. Furthermore, notice that MultiMob waits 40 s before opening the cellular subflow using `FJI`, when the receive timer detects that no more data is received after some time without having received a `MP_IDLE`. Based on our model [20], NBK consumed 444 J for the cellular interface during the test (1386 mW), while MultiMob spent 329 J (1028 mW).

In the BK vs. MultiMob test, the network interface usage is similar, i.e., WiFi is used when available. Over a dozen of runs we observed no stall, except for a test that impacted both Backup and MultiMob. The buffer playing time at client side for that test is shown on Fig. 11. At 50 s (first **C**-**D** pass), MultiMob faces an half-second stall time, due to the reception of a packet on the WiFi network while the cellular subflow was already established. Since packets are acknowledged on the subflow they came from, the `MultiMob` server-side scheduler then tries to reply on the WiFi subflow, but it was meanwhile lost. After facing a RTO, the server reinjects this reply on the cellular subflow and the connection continues. The BK case experienced a 3 s stall time at time 205 s (second **C**-**D** pass). This stall was caused by the `default` scheduler that favors the WiFi subflow over the backup subflow on the cellular interface. Indeed, after 200 s, the WiFi was underperforming, the server experienced RTOs and reinjected data on the cellular subflow, but it then came back. When the WiFi signal eventually disappeared, the RTO value increased because of previous losses and the RTO expired seconds after the actual WiFi loss. Again, the BK case opened the cellular path at the beginning of the connection, while this happened at 45 s by MultiMob. Furthermore, MultiMob has a smaller cellular energy consumption with 319 J (994 mW), though the BK one remains close with 347 J (1083 mW).

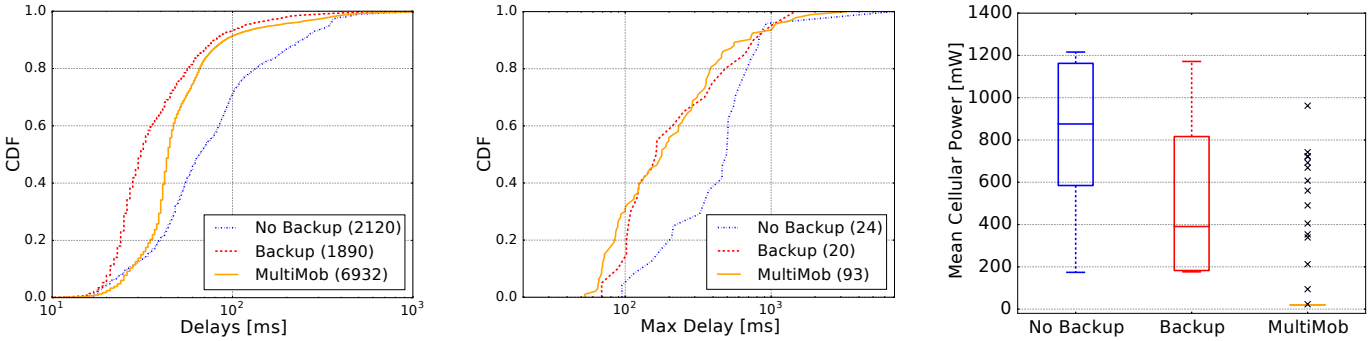
B. Measurements with Real Users

This section summarizes active measurements performed on Nexus 5 devices distributed to a few students and academics over a period of seven weeks (28th January - 22nd March 2017). We installed on each smartphone an Android application that periodically changes the network configuration, either once during night or after a reboot. Our measurement application runs in the background and sends data when it detects that the smartphone moves. Network conditions of tests depend on the presence of WiFi and/or LTE networks. To observe the performance of MultiMob to switch from the WiFi network to the cellular one, we only consider here tests where both WiFi and cellular interfaces are online at the beginning of the tests. Notice that WiFi can be lost during some tests.

Figure 12a shows that nearly all simplified interactive requests are answered within one second. Notice that simultaneously using two paths for such traffic as with the NBK can lead to increased response delays because of network heterogeneity between paths. Figure 12b plots the maximum delay observed during the tests. For all configurations, the maximum delay observed to answer simplified interactive requests remains within one second, with rare outliers higher than two seconds. Though the oracle detection to trigger cellular subflow is the largest delay component, MultiMob does not impact too much the request traffic when WiFi is lost. The main difference between MultiMob and both NBK and BK resides in scenarios where the WiFi remains alive during the whole test. The energy consumption computed on the entire traffic collected during the test is shown in Fig. 12c. Since NBK and BK always open additional subflows at the beginning of the connection, they consume energy, even if no real data is sent on that interface. Background connections initiated by real users can sometimes increase energy consumption by using the cellular interface. In contrast, since most of the time MultiMob does not create additional subflows, its cellular energy consumption is very low. MultiMob can thus keep low latency for delay-sensitive applications while limiting energy impact of Multipath TCP.

VI. Conclusion

Given that smartphones have both cellular and WiFi interfaces, users expect them to be able to perform seamless



(a) All request delays. The legend contains number of requests for the configuration.

(b) Maximal delays. The legend shows the number of tests for the configuration.

(c) Estimated mean cellular power [20] during the test when primary path stays alive.

Fig. 12: Simplified interactive traffic with real users (easier to see with color).

handovers between those two network interfaces. Multipath TCP enables such seamless handovers since it can use both cellular and WiFi interfaces for a single connection. Using both interfaces simultaneously is too expensive from an energy viewpoint. We propose, implement and evaluate MultiMob, a set of improvements to the Multipath TCP implementation and protocol. MultiMob uses *break-before-make* to minimise energy consumption. It extends Multipath TCP to support immediate retransmissions over a different interface. Furthermore, thanks to its scheduler, a server automatically selects the best performing interface to respond to requests from a smartphone. Our measurements indicate that MultiMob improves the performance of Multipath TCP on smartphones while minimizing energy consumption.

MultiMob is available: http://multipath-tcp.org/multimob

References

[1] Cisco visual networking index, February 2010.

[2] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: A measurement study and implications for network applications. In *IMC'09*, pages 280–293. ACM, 2009.

[3] Olivier Bonaventure and SungHoon Seo. Multipath TCP deployments. In *IETF Journal*, volume 12, pages 24–27. November 2016.

[4] Bob Briscoe et al. Reducing internet latency: A survey of techniques and their merits. *IEEE Communications Surveys & Tutorials*, 18(3):2149–2196, 2014.

[5] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *USENIX'10*, volume 14. Boston, MA, 2010.

[6] Yung-Chih Chen et al. A measurement-based study of MultiPath TCP performance over wireless networks. In *IMC'13*, pages 455–468. ACM.

[7] Stuart Cheshire et al. Advances in networking, part 1. https://developer.apple.com/videos/play/wwdc2017/707//, June 2017.

[8] Quentin De Coninck, Matthieu Baerts, Benjamin Hesmans, and Olivier Bonaventure. Observing real smartphone applications over Multipath TCP. *IEEE ComMag*, 54(3):88–93, March 2016.

[9] Quentin De Coninck, Matthieu Baerts, Benjamin Hesmans, and Olivier Bonaventure. A first analysis of Multipath TCP on smartphones. In *PAM'16*, pages 57–69. Springer, 2016.

[10] Quentin De Coninck and Olivier Bonaventure. Every millisecond counts: Tuning Multipath TCP for interactive applications on smartphones. Technical report. Available at http://hdl.handle.net/2078.1/185717.

[11] Shuo Deng et al. Wifi, lte, or both? measuring multi-homed wireless internet performance. In *IMC'14*, pages 181–194. ACM, 2014.

[12] N Dukkipati et al. Tail Loss Probe (TLP): An algorithm for fast recovery of tail losses. *IETF Draft, draft-dukkipati-tcpm-tcploss-probe-01*, 2013.

[13] Hossein Falaki et al. Diversity in smartphone usage. In *MobiSys'10*, pages 179–194. ACM, 2010.

[14] Hossein Falaki, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin. A first look at traffic on smartphones. In *IMC '10*, pages 281–287, New York, NY, USA, 2010. ACM.

[15] Simone Ferlin et al. Blest: Blocking estimation-based mptcp scheduler for heterogeneous networks. In *IFIP Networking Conference (IFIP Networking) and Workshops, 2016*, pages 431–439. IEEE, 2016.

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

[17] Bo Han et al. Mp-dash: Adaptive video streaming over preference-aware multipath. In *CoNEXT'16*, pages 129–143. ACM, 2016.

[18] Bo Han, Feng Qian, Shuai Hao, and Lusheng Ji. An anatomy of mobile web performance over Multipath TCP. In *CoNEXT '15*, pages 5:1–5:7, New York, NY, USA, 2015. ACM.

[19] Nikhil Handigol et al. Reproducible network experiments using container-based emulation. In *CoNEXT'12*, pages 253–264. ACM, 2012.

[20] Junxian Huang et al. A close examination of performance and power characteristics of 4g lte networks. In *MobiSys'12*, pages 225–238. ACM.

[21] Will Knight. Conversational interfaces. *MIT Technology Review*.

[22] Ming Li et al. Multipath transmission for the internet: A survey. *IEEE Communications Surveys Tutorials, vol. PP*, (99):1–41, 2016.

[23] Yeon-sup Lim et al. How green is Multipath TCP for mobile devices? In *All Things Cellular'14*, pages 3–8. ACM, 2014.

[24] Yeon-sup Lim et al. Design, implementation, and evaluation of energy-aware Multi-Path TCP. In *CoNEXT'15*, page 30. ACM, 2015.

[25] Yeon-sup Lim et al. Ecf: An mptcp path scheduler to manage heterogeneous paths. In *CoNEXT'17*, pages 33–34. ACM, 2017.

[26] Ana Nika et al. Energy and performance of smartphone radio bundling in outdoor environments. In *WWW'15*, pages 809–819. ACM, 2015.

[27] Ashkan Nikravesh et al. An in-depth understanding of Multipath TCP on mobile devices: Measurement and system design. In *Mobicom'16*, pages 189–201. ACM, 2016.

[28] Bong-Hwan Oh and Jaiyong Lee. Constraint-based proactive scheduling for mptcp in wireless networks. *Computer Networks*, 91:548–563, 2015.

[29] Christoph Paasch, Sebastien Barre, et al. Multipath TCP in the linux kernel. http://www.multipath-tcp.org, 2017.

[30] Christoph Paasch et al. Exploring mobile/wifi handover with Multipath TCP. In *CellNet'12*, pages 31–36. ACM, 2012.

[31] Christoph Paasch et al. Experimental evaluation of Multipath TCP schedulers. In *CSWS'14*, pages 27–32. ACM, 2014.

[32] Christopher Pluntke et al. Saving mobile device energy with Multipath TCP. In *MobiArch'11*, pages 1–6. ACM, 2011.

[33] Sivasankar Radhakrishnan et al. TCP Fast Open. In *CoNEXT'11*, page 21. ACM, 2011.

[34] Costin Raiciu et al. Improving datacenter performance and robustness with multipath tcp. In *CCR'11*, volume 41, pages 266–277. ACM, 2011.

[35] Costin Raiciu et al. Opportunistic mobility with Multipath TCP. In *MobiArch'11*, pages 7–12. ACM, 2011.

[36] Costin Raiciu et al. How hard can it be? designing and implementing a deployable Multipath TCP. In *NSDI'12*, pages 29–29, 2012.

[37] SungHoon Seo. Kt's giga lte. IETF 93, 2015.

[38] Hassan Sinky et al. Proactive Multipath TCP for seamless handoff in heterogeneous wireless access networks. *IEEE Transactions on Wireless Communications*, 15(7):4754–4764, 2016.