Every Millisecond Counts: Tuning Multipath TCP for Interactive Applications on Smartphones

Quentin De Coninck*
UCLouvain, Belgium
quentin.deconinck@uclouvain.be

Olivier Bonaventure
UCLouvain, Belgium
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.

In this paper, we explore different techniques to tune Multipath TCP for interactive applications. On smartphones, user experience is always a compromise between network performance and energy consumption or battery lifetime. We first explore the open-source Multipath TCP implementation in the Linux kernel. Our measurements indicate that Linux needs to better take energy consumption into account. We then propose, implement and evaluate MultiMob, a solution providing fast handover with low energy consumption 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.

1 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 [25, 29]. The user uses his/her voice instead of buttons to interact with the application that sends voice samples to the cloud. For these applications, latency is key and many protocols have been tuned during the last years to reduce their latency [9, 42, 46].

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 fewer energy [28, 33]. 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 users only use their cellular interface for voice calls and when there is no WiFi network available [14].

Multipath TCP is a recent TCP extension [20] that was designed with these smartphones in mind. Pluntke et al. [40] and then Raiciu et al. [44] first discussed the expected benefits of Multipath TCP on such mobiles devices. Later, Paasch et al. implemented handover features [37] in the Linux kernel [36]. Lim et al. showed the importance of taking energy consumption into account [31].

Industry has already adopted Multipath TCP on smartphones with two major deployments [8]. Apple uses Multipath TCP for its Siri voice activated application since 2013 and has recently announced that any application will be able to use Multipath TCP on iOS11 [14] that will be released in fall 2017. This is the largest deployment of Multipath TCP today [8] 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 [47].

Many authors studied and tuned the bandwidth aggregation capabilities of Multipath TCP [12, 16, 22, 30, 34, 45] in mobile networks. Although popular in the scientific literature, this use case is not the main use case for Multipath TCP on smartphones [14]. Smartphones rarely exchange large files [15] that would benefit from bandwidth aggregation. Measurement studies [15, 19] and Sect. 2.1 show that smartphones mainly use short TCP connections. This is confirmed by Apple that will open Multipath TCP on iOS11 to provide seamless handovers and support interactive and voice-activated applications [14].

We first describe the current state-of-the-art of Multipath TCP on smartphones in Section 2. Section 3 proposes a simple benchmark to analyze the network interactions of voice activated applications based on passive measurements collected with a popular voice activated application. We then propose MultiMob, a series of improvements that adapt Multipath TCP and its reference implementation in the Linux kernel to

*FNRS Research Fellow

the requirements of today's smartphone applications. More precisely, MultiMob provides a good compromise between latency and energy consumption.

A Multimob server replies on the subflow used by the smartphone (§ 4.1). If a smartphone sends a request over a cellular subflow because its WiFi subflow performs badly, the server should reply over the same subflow since the server has no information about the client's wireless conditions.

MultiMob minimizes cellular usage and unused subflows (§ 4.2). Like the forthcoming iOS11 [14], MultiMob prefers to use the WiFi interface over the cellular interface on smartphones. MultiMob replaces the *make-before-break* strategy of the Multipath TCP implementation on Linux by *break-before-make*. With this strategy, the cellular interface is only used after a failure of the WiFi interface.

MultiMob limits handover delays (§ 4.3). 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 handshake.

In Sect. 5, we evaluate MultiMob in various network conditions and different applications. Finally, Sect. 6 concludes and provides information required to reproduce the results described in this paper.

2 STATE OF THE ART AND MOTIVATION

Multipath TCP [20] was designed with multihomed devices such as smartphones in mind. It enables them to exchange data belonging to a single connection over different paths or network interfaces. Multipath TCP is described in details in [20, 45]. We only briefly summarize its main features here. A Multipath TCP connection is in fact a combination of different TCP connections, called *subflows* in [20], 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 key exchanged during the initial handshake [20]. Data can be exchanged over the initial subflow and both the client and the server can create additional subflows to use other paths or because of 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 (DSS) 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 subflow failed or became unresponsive. *Reinjecting* data over a different subflow is key to support handovers [37, 45].

There are two main implementations of Multipath TCP on mobile nodes: Apple's implementation on iOS [14] and the open-source Linux implementation [36]. We focus on the latter because it fully implements the protocol and can be easily modified. Besides supporting all the features described in [20], it includes several heuristics that are important for performance [45], but are not specified in [20] since they do not impact interoperability.

A first component of the Multipath TCP implementation in the Linux kernel is the *path manager* that resides in the Linux kernel. It determines when additional subflows must be established. The initial subflow is always established on the interface that points to the current default route. The default path manager is called `fullmesh` [36]. On a client, it 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 plays an important role in the operations of Multipath TCP when there are several subflows [38]. 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. Other schedulers have been proposed [21, 35].

Multipath TCP [20] 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 the first public beta release of iOS11 [14], released one week before the submission deadline, also sets the backup bit on the cellular subflow to discourage the utilization of the cellular interface to transport data. Early versions of the Linux Multipath TCP implementation (up to version 0.89) considered that a subflow had failed once its state was removed, either due to the failure of the corresponding interface or after the reception of a RST packet. Unfortunately, some subflows fail slowly, e.g., when TCP tries to retransmit the same data seven times and doubles its retransmission timer between two retransmissions on a bad WiFi network. This behavior changed in version 0.90 that introduced an "improved backup mode" [2]. With this modification, a subflow is considered to be in a

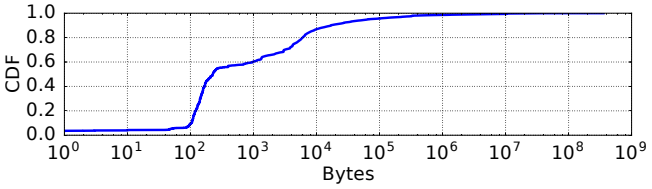


Figure 1: Number of bytes carried by observed Multipath TCP connections.

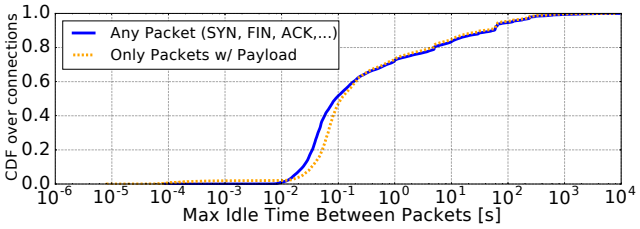


Figure 2: Measured maximum idle time on smartphone connections carrying data.

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. Starting from version 0.90, the default scheduler uses a backup subflow if all the regular subflows are in the *potentially failed* state.

2.1 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 measurements with friendly users.

Smartphone applications rarely perform bulk transfers Multipath TCP was designed to aggregate bandwidth and many articles have evaluated whether Multipath TCP reaches that objective [12, 16, 38, 39, 43]. However, smartphones rarely exchange very long files [15, 18]. To confirm this, we mimic the Multipath TCP deployment in Korea and deploy a SOCKS server in the cloud and distributed 6 Nexus 5 smartphones patched with Multipath TCP on Android 6.0.1 to actual users. Both the smartphones and the server were configured with scripts that collect packet traces and other statistics. Our spans two months and contains 53,645 connections. Fig. 1 shows the CDF of the number of bytes exchanged over the captured TCP connections. Around 90% of them carry less than 10 KB. This confirms earlier results [15, 18]. Another important point that we noticed about the TCP connections used by smartphones is that many connections experience large idle times. Figure 2 provides CDFs of the idle times observed on the captured connections. It shows that most of the connections experience idle times of less than one second. However, 15% of the connections experience more than 10 seconds of idle time between two consecutive data packets. From TCP's point of view, an idle TCP connection is not a problem. However, from an energy viewpoint an idle connection can consume energy if the radio needs to remain active to support this idle connection.

Many subflows do not carry data On our smartphones, the Linux Multipath TCP `fullmesh` path manager immediately creates subflows on all active interfaces. Our passive measurements confirm this. We observe that 90% (resp. 95%) of the first additional subflows are established within the first 100 ms (resp. 1 s) of the connection. Unfortunately, most of these subflows are useless. Around 90% of the captured Multipath TCP connections do not send *any* data over the additional subflows, i.e., all data is exchanged on the initial subflow. With the `default` scheduler, if the initial subflow exhibits a lower round-trip-time than the additional ones, Multipath TCP will only use the initial one. Many WiFi networks, including ours, have lower round-trip-times than the cellular networks. Furthermore, previous studies indicate than Multipath TCP can even perform worse than regular TCP on short flows in heterogeneous networks [22, 34].

Mismatch with user expectations Most users favor WiFi over cellular for both monetary and power consumption [14, 28] reasons. 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 scheduling decision to send the next packet over a particular subflow is taken by the sender of the packet. In practice, smartphones mainly receive data [15, 18], meaning that most of the scheduling decisions are taken by remote servers. Unfortunately, the only metric that a Multipath TCP server uses to determine the best subflow to send data is the measured round-trip-time. However, with the decreasing latency of LTE and network operators providing WiFi access, the cellular network can sometimes exhibit a lower latency than the WiFi one [49]. The server scheduling decision can then go against the user expectations.

Backup subflows consume too much energy According to the Multipath TCP specification, one way to minimize the utilization of the cellular network is to always establish the cellular subflow as a backup subflow [20, 31]. 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, 10, 14, 19, 51], and many works explore various solutions to limit the battery usage on such devices [11, 24, 41, 50, 52]. However, opening a subflow on the cellular interface without using it is expensive from an energy consumption viewpoint [16], the WiFi interface consuming at least five times less than the LTE one [33]. In the

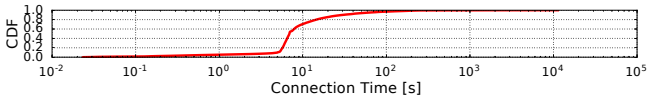
remaining of this paper, we use the LTE model described in [28] to estimate the cellular power consumption (we expect similar results with other models like [33]). Based on this model, we identify user-initiated connections in Multipath TCP Backup mode that creates cellular subflow without sending any data on it. On overall, 12,274 connections created 626 RRC_CONNECTED periods on cellular and spent more than 4500 J just because of SYN sent. In the model used [28], for a smartphone, opening a single cellular subflow 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 [37] is thus very expensive from an energy consumption viewpoint. Our early experiments confirm that the first public beta of Apple iOS11 does not create cellular subflows immediately.

Related Works Lim et al. [32] 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. Furthermore, the code does not seem publicly available. Sinky et al. [48] 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.

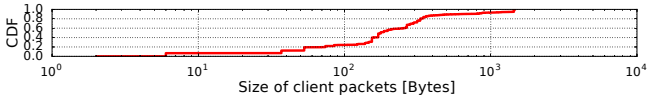
3 VOICE ACTIVATED APPLICATIONS

Voice activated applications will likely play a growing role on smartphones [25, 29]. One of them is Apple's Siri assistant. Despite being the first application using Multipath TCP at large scale [8], little is known and studied about the Siri traffic. Assefi et al. analyze the reaction of Siri to losses and jitter in [5] but do not provide a model of the traffic produced by this application. Siri interacts with a few well-known servers. We captured all the packets exchanged with these servers from a university campus network serving several thousands of wireless devices during a week. We focus our analysis on the 18, 166 Multipath TCP connections that did not perform any handover to a cellular network since we were unable to capture packets over that network.

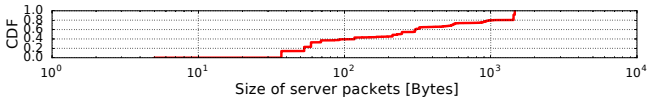
Siri runs over HTTPS, probably to ensure privacy and also reduce the risk of middlebox interference. The client sends requests that are immediately answered by the server. Fig. 3 presents some statistics extracted from the Siri trace. 60% of the Siri connections last between 5 and 10 seconds. This time can probably be correlated to the time needed to ask a question to the assistant and getting back the reply. Notice that 15% of the connections last more than 20 seconds, probably because of successive user interactions with the Siri application. Figure 3b shows that 77% of the data packets sent by Siri clients have a size between 50 and 500 bytes. Only 7% of



(a) Duration of the TCP connections created by Apple's Siri.



(b) Size of the data packets sent by Siri clients.



(c) Size of the data packets returned by Siri servers.

Figure 3: Analysis of the Siri traces collected on campus WiFi network.

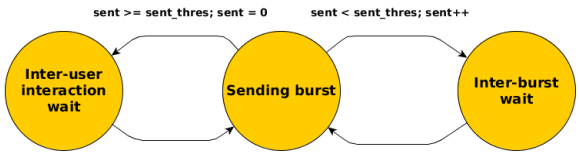


Figure 4: State machine of a simple voice activated application.

packets sent by the clients are longer than 1000 bytes. Those packets are often sent in bursts of different sizes, probably related to the voice sampling process. In comparison, MSS-sized packets represent only 20% of the packets sent by the server. The large fraction of short packets suggests that the Siri application uses the NO_DELAY socket option to disable the Nagle algorithm.

To evaluate the interactions between a simple voice-activated application and Multipath TCP, we use a simplified model of the behavior of such an application. Our model is a three-states process shown in Fig. 4. 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 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 runs for 80 seconds and measures the delay between each request and the server's response.

Algorithm 1 isNewerThan(sf, osf)

Input: sf a subflow, osf another subflow or None
Output: Return true if sf

```
1: if osf is None then return true
2: if sf has newer original reception than osf then
3:     return true                    ▷ See text for definition
4: return false
```

Algorithm 2 `MultiMob`'s server-side scheduler tracks the subflow that was most recently used by the client

Output: $bestsf$ next subflow to use

```
1: bestsf ←None
2: for sf in subflows do
3:     if isNewerThan(sf, bestsf) then        ▷ See Alg. 1
4:         if sf is available then
5:             bestsf ← sf
6: return bestsf
```

4 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.

4.1 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. This scheduler works well in a variety of environments [38]. However, selecting subflows only on the basis of their round-trip-time 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 round-trip-time. 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 operation of the server-side scheduler is described in Algorithm 2. It 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 and 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.

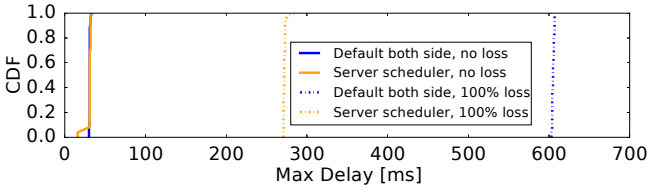


Figure 5: Maximum delay to return an answer to simplified interactive requests. Each scenario ran 25 times. Losses begin to occur when the client is in *inter-user interaction wait* state, i.e., between request bursts.

We evaluate the MultiMob scheduler using Mininet [23] in a simple topology with two disjoint paths between the client and the server. Simulations are based on Multipath TCP v0.91 in Linux 4.1. 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 5 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 will experience an RTO before reinjecting the packet on the additional subflow. However, with the `default`

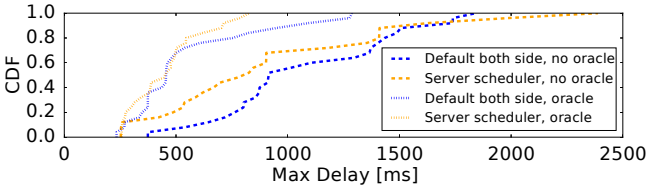


Figure 6: Maximum delay to answer simplified interactive requests. Each configuration was run 25 times. Random losses of 25% occur once the connection is established.

IP_{src}	IP_{dst}	Net interf.	TCP sfs	TCP stats
1.2.3.4	4.5.6.7	WiFi	[tp_1, tp_3]	sloss 2%, ...
2.3.4.5	5.6.7.8	Cellular	[tp_2]	sloss 0%, ...
2.3.4.5	4.5.6.7	Cellular	[tp_4, tp_5]	sloss 15%, ...

Table 1: Oracle monitoring table example.

scheduler, the server does not 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, hence reduces the delay perceived by the application. Notice that when the network is not completely broken, the performances of both `default` and `MultiMob` schedulers are less predictable, as shown on Fig. 6. The default scheduler still prefers the low RTT subflow (even if lossy), and the path selected by the `MultiMob` server-side scheduler is function of the last received packet. To cope with this problem, the client should terminate the lossy subflows.

4.2 Break-Before-Make

The Multipath TCP implementation in the Linux kernel uses *make-before-break* to manage subflows. When a Multipath TCP connection starts, the `fullmesh` path manager opens the connection over the primary interface and then 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. From an energy viewpoint, the smartphone's cellular interface consumes almost the same energy when only SYN and FIN packets are exchanged and when additional data packets are exchanged. Indeed, as discussed in Sect. 2.1, subflow establishments can be spaced such that the LTE interface remains in `RRC_CONNECTED` for a long period of time, even if no data packets are sent on the cellular interface.

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 will experience retransmissions and losses, even if they belong to different connections. To track those events, our oracle maintains a monitoring table of netpaths as shown on Tab. 1. A *netpath* is a tuple (IP_{src}, IP_{dst}, network interface). We aggregate the information on a per network flow basis instead of on a per transport flow basis to reduce the size of the monitoring table. This structure is well adapted to deployments with SOCKS proxies such as [47] where all Multipath TCP connections are terminated on the proxy.

When a new subflow is created, the oracle checks if its corresponding netpath exists. If so, the subflow is added to the list of subflows matching this entry. Otherwise, a new entry is created. When a subflow stops, it is removed from the subflow list it belongs to. If no more subflows are associated to a netpath, the entry is removed from the monitoring table.

Our oracle computes every T_s seconds statistics based on the subflows associated to a given netpath. T_s is empirically set at 500 milliseconds, a discussion about this value is provided later. 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 [17]. 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). These V-EMA reduce to the three following equations

$$val_{i+1} = \frac{prod_{i+1}}{vol_{i+1}} \tag{1}$$

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

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

where val_{new} is the new value of the studied metric (e.g., number of 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. Note that $prod_0 = vol_0 = val_0 = 0$ and no value is computed if vol_{new} is 0. $\alpha \in [0, 1]$ is a tuning parameter, experimentally set to 0.5.

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. Thanks to our oracle, backup subflows are immediately created for all established subflows once a problem has been detected.

To assess the benefits of the oracle and determine the threshold value for *sloss*, we still rely on Mininet simulations with simplified interactive requests while a light continuous traffic is present. Figure 7a shows the maximal latency to answer a simplified voice activated request and Fig. 7b 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 [28], 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 triggers backup subflow establishments 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 is faced. On the contrary, the simplified voice activated 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%), the no oracle case tries to reuse the lossy path once some ACK manage to reach the host, while the oracle prevents this behavior. With the oracle the establishment of the additional subflow depends on the network conditions and the *sloss* threshold parameter. When set to a low value like 10%, max 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, a *sloss* threshold of 25% seems to be a good 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 trying to use a subflow that might hurt interactive applications because of lack of retransmission reactiveness.

Notice that the reactivity of the oracle also depends on the oracle timer T_s. Indeed, as shown on Fig. 8, the lower T_s, the quicker is 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 spends lot of CPU time to compute its monitoring table. The value of 500 ms for T_s seems to be a good trade-off between sub-second reactivity and low CPU usage on mobile devices.

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. A naive approach would be to let the client regularly probe the server, with keep-alives, but active probing consumes energy and can degrade wireless performances [27] and is thus discouraged.

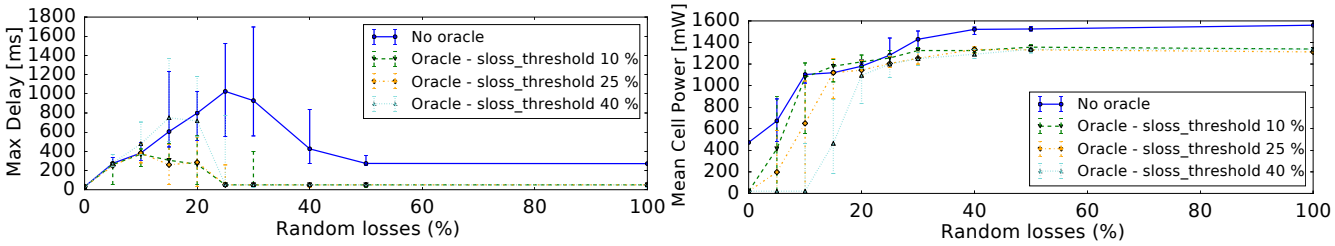
There are different ways to cope with this problem. A Multipath aware HTTP client could use the `Content-Length` header sent by the server to detect idle times in the middle of a transfer and create an additional subflow at that time, e.g. via the enhanced socket API [26]. Another approach is to extend Multipath TCP. This can work with existing applications. We modify Multipath TCP so that the server can indicate to the client that a data transfer is not yet finished. 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.

To evaluate this solution, we perform a Mininet simulation where the client downloads a 20 MB file and we add 100% losses on the primary path after 1.5 s. Fig. 9 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.

Shortly before the submission of this paper, Apple announced the first public beta of iOS11 [14]. This release includes WiFi Assist that detects badly performing wireless networks and apparently triggers the creation of backup subflows. Unfortunately, Apple did not provide enough details to enable use to compare our oracle with WiFi Assist.

4.3 Immediate reinjections

The *break-before-make* approach described in the previous section is beneficial from an energy viewpoint. However, a



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

(b) Estimated mean cell power consumption.

Figure 7: 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.

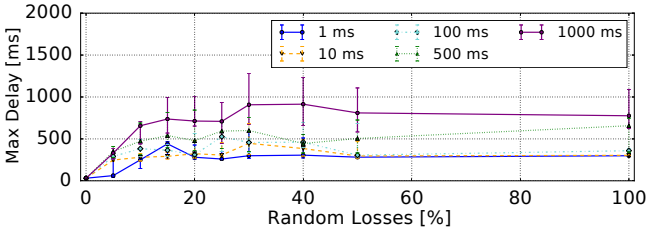


Figure 8: Varying T_s for interactive traffic, with *sloss* set to 25%. Showing 25th, 50th and 75th percentiles.

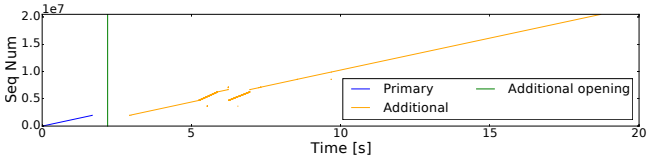


Figure 9: Time-sequence graph of the packets received by a client during a 20 MB HTTP GET. The primary subflow suffers from 100% losses at 1.5 s.

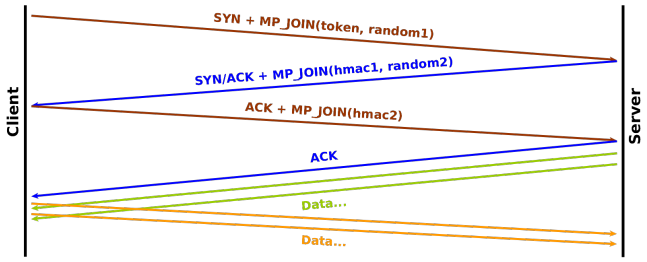
mobile typically detects the failure of a wireless interface by the expiration of its retransmission timer or TLP probe. This unacknowledged data can only be retransmitted over another interface once a subflow has been established over this interface. Multipath TCP [20] requires a four-way handshake 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. 10a). Unfortunately, this handshake delays the reinjection of the lost data since the client cannot send data before having received the fourth ACK [20].

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` and `FAST_JOIN_OUT`. This is different than TCP Fast Open [13, 42] because a new subflow is established between hosts that already share state for one Multipath TCP connection.

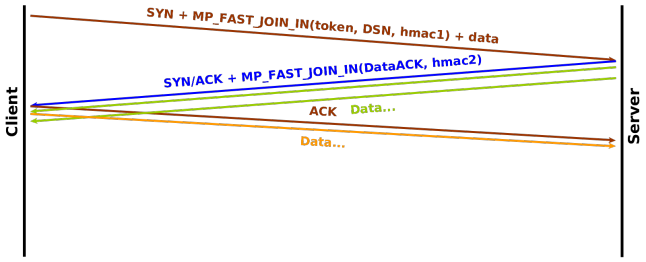
A naive approach would be to simply place data inside the SYN and require the server to accept this data immediately. Unfortunately, this solution would cause security problems because this SYN is not authenticated. The `MP_JOIN` option that it carries contains only the 32 bits token that identifies the connection and a random number that is used to authenticate the server (Fig. 10a). 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 [6], e.g. on an open WiFi network.

The `FAST_JOIN_OUT` option described on Fig. 10b solves this problem and allows the client to carry data in the initial SYN without causing any security issue. Our `FAST_JOIN_OUT` 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 `FAST_JOIN_OUT` option for a given DSN. To cope with lost acknowledgments, if *EDSN* is the next expected DSN on the server and *SDSN* 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 `FAST_JOIN_OUT` 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



(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.

Figure 10: Time-sequence diagrams for the establishment of additional subflows.

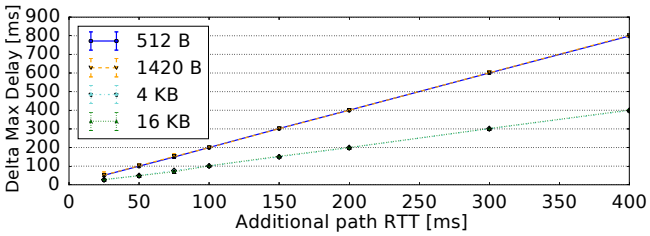


Figure 11: Delta of max delay between normal and fast joins depending on the request size. Markers are medians over 6×2 runs, bars show min and max.

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 `FAST_JOIN_IN` option (not shown for space limitations). This option is very similar to the `FAST_JOIN_OUT` 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. The client still has to wait for the SYN+ACK before sending data. By using the `FAST_JOIN_IN` option, the data transfer can resume after 1 RTT, instead of 2 RTTs with normal join.

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 11 shows the difference of the delays of the first request following the loss event using normal and fast joins. To limit the variability of the detection of the loss event between normal and fast joins, the oracle timer T_s is set to 1 ms. 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, if the server pushes data.

5 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. Four configurations are studied: 1) *No backup*, MPTCP v0.89 with the `fullmesh` path manager; 2) *Backup*, MPTCP with current Linux backup behavior on cellular; and 3) *MultiMob*, the proposed solution described in Sect. 4. We use two servers. The first one, configured with the `default` scheduler, is used by vanilla Multipath TCP configurations. The second one, configured with the `MultiMob` server-side scheduler, is used by MultiMob. In this section, we first explore particular usecases with micro-benchmarks to understand the benefits of MultiMob. We then compare at a larger scale vanilla Multipath TCP with MultiMob through active measurements performed on a set of modified Android 6 devices used by real users.

5.1 Micro-Benchmarks

We first compare the standard Android 6.0.1 network handover with MultiMob. We then study actual user mobility with both simplified interactive and streaming applications.

5.1.1 Android Network Handover. Android 6.0.1 includes software that controls the network interfaces. As iOS11, Android favors WiFi over cellular and sets the WiFi interface as the default one. Once connected to a WiFi interface, Android stops the cellular interface. We modify Android so that the cellular interface remains always on. To avoid staying on underperforming WiFi networks, Android 6 monitors the WiFi interface and computes layer-2 transmission and reception success rates with a V-EMA [3]. Android then computes a network score that combines these success rates, the RSSI and the WiFi bandwidth. If the WiFi score drops below the one of the cellular, Android selects the cellular network as the default one and tears down the WiFi network.

When an Android smartphone associates to a WiFi access point (AP), it first sends a HTTP request to a Google server

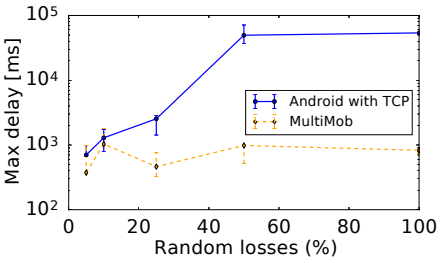


Figure 12: Max delays for simplified interactive (without inter-request wait time) over 3 runs, showing minimal, median and maximal values.

to detect if the AP is behind a captive portal or not [3]. If this request succeeds, the WiFi interface is considered as functional. Android does not seem to monitor the performance of the selected WiFi network besides tracking the WiFi beacons. This can cause degradations with overloaded shared WiFi APs [4]. In such networks, priority is given to the network subscriber, the remaining capacity being shared among the other users. If the network subscriber makes intensive usage of his network, the community users might experience consequent losses. To confirm this we connected two phones to a WiFi access point providing Internet access. Both run our simplified interactive application without inter-user interaction wait during 80 s. One smartphone uses TCP and the other MultiMob. After 5 s, the WAN interface of the WiFI AP starts to experience random losses on outgoing packets during 50 s. Figure 12 shows the maximum delay experienced by the simulated interactive client depending on the percentage of random losses on the WAN interface. With low loss percentages ($\leq 10\%$), both phones experience similar delays. This is because losses can be recovered with fast retransmits and the loss rate is not sufficient for MultiMob to use cellular subflows. However, with higher loss rates, while MultiMob switches to the cellular network, the Android insists on using the WiFi network and does not break the TCP connection. Indeed, the WiFi performance remain good for Android because WiFi beacons are successfully transmitted, even if the Internet connectivity is completely congested. This test shows that MultiMob, initially single-path, can react to such failures while Android continues to use underperforming networks.

5.1.2 Mobility Scenarios. To evaluate how MultiMob performs in changing wireless conditions, we go for a short walk (Figure 13) 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

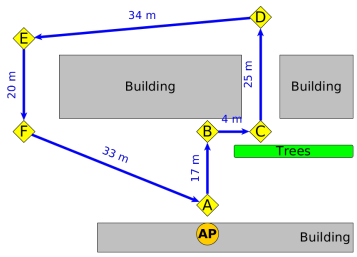
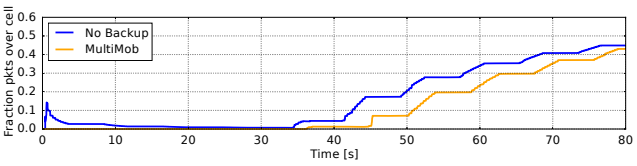
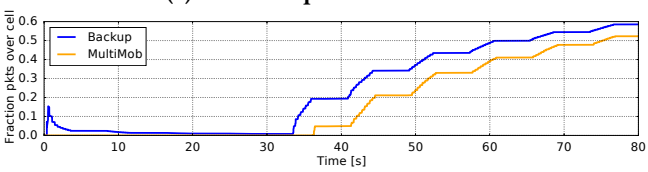


Figure 13: Walk map for micro-benchmarks.



(a) No Backup vs. MultiMob.



(b) Backup vs. MultiMob.

Figure 14: 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	1187	100	657

Table 2: Aggregated results from simulated voice activated micro-benchmarks. MultiMob shows the median value over runs. MD = Max Delay, RA = Requests Answered, CP = mean Cell Power consumption.

location **D**. Starting at location **F**, the WiFi signal becomes available again.

Simulated Interactive Traffic Our test phones send 100 requests during our 80 s walk from **A** to **D**. Figure 14 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. 2 shows aggregated results related to these tests. In the vanilla Multipath TCP cases, 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 good results

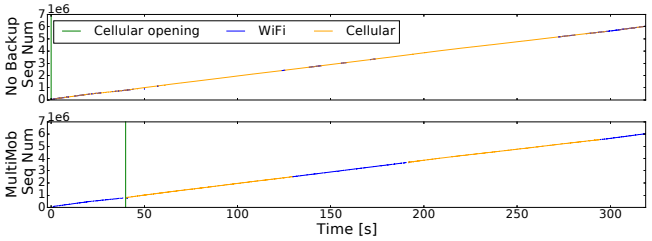


Figure 15: Time-sequence graph of the server streaming flow as perceived by the client for the No Backup vs. MultiMob. Color indicates on which interface packet was received.

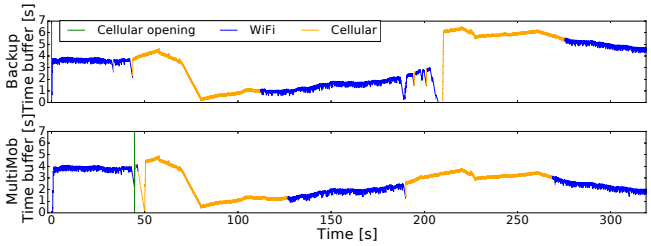


Figure 16: Playing time of the client buffer for the worst case in Backup vs. MultiMob test. Color indicates on which interface packet was last received.

of the No Backup case. 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 No Backup and Backup cases often experience a lower maximal delay than MultiMob. Since the cellular subflow is already established, the No Backup and Backup 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 No Backup and Backup cases[1]. On the opposite, MultiMob consumes less cellular energy since it delays the utilization of the cellular interface.

Streaming Traffic For this test, we configure the smartphones to stream a web radio over HTTP while performing twice the walk presented on Fig. 13. Our servers relay the same web radio at fixed bitrate using Icecast. Since all the data flows from the server to the client all the scheduling decisions are made by the server.

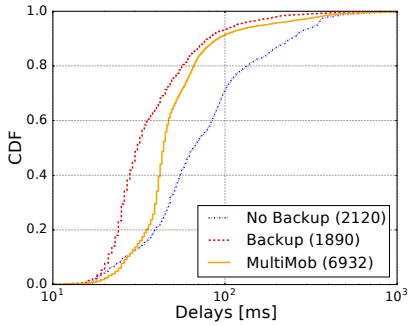
Figure 15 shows the time-sequence graph for the No Backup vs. MultiMob test. We did not observe any stall during those experiments, which is important for a streaming application. However, the No Backup case sends data nearly exclusively on the cellular interface, even when the WiFi network is available. From the server perspective, the cellular network appears to be more stable with an often lower estimated smoothed RTT than the WiFi one. 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, when the receive timer detects that no more data is received after some time without having received a `MP_IDLE`. Based on our model, No Backup consumed 444 J for the cellular interface during the test, while MultiMob spent 329 J.

In the Backup vs. MultiMob test, the distribution between the WiFi and cellular is similar, i.e., WiFi is used when available. Over a dozen of runs we did not observe any 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. 16. 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 Backup 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 Backup case opened the cellular path at the beginning of the connection, while it was done at 45 s by MultiMob. Furthermore, MultiMob has a smaller cellular energy consumption with 319 J, though the Backup one remains close with 347 J.

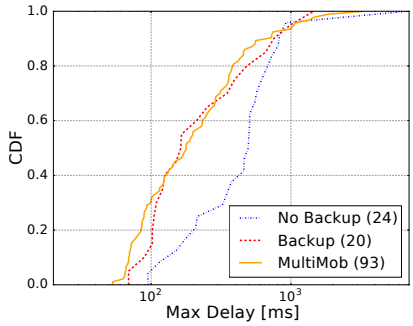
5.2 Measurements with Real Users

This section summarizes active measurements performed on Android devices distributed to users over a period of one month. 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 can vary depending on the presence of WiFi and/or

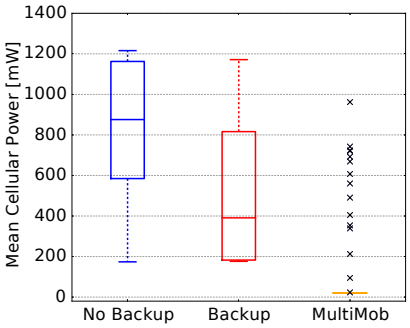
[1]The delay is actually very dependent of the wireless conditions of a particular run. The lowest max delay observed for MultiMob over runs was 599 ms.



(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 during the test when primary path stays alive.

Figure 17: Simplified voice activated traffic with real users (easier to see with color).

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 measurements. Notice that WiFi can be lost during some tests.

Figure 17 shows measurements with the simplified voice activated traffic. Figure 17a shows that nearly all requests are answered within one second. Notice that simultaneously using two paths for such traffic as with the No Backup can lead to increased response delays because of network heterogeneity between paths. Figure 17b 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. This shows that with MultiMob, the request traffic is not too impacted by the delaying of the cellular subflow establishment when the WiFi is lost. The main difference between MultiMob and vanilla MPTCP configurations 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 on Fig. 17c. Since vanilla MPTCP configurations 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.

6 CONCLUSION

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

Name	Language	Space	LoC
`MultiMob` scheduler	C	Kernel	50
Oracle	C	Kernel	600
Fast Join	C	Kernel	~ 1000
Control App	Java	Android	4500
Measurement App	Java	Android	2200

Table 3: Code contributions of this paper.

REPRODUCIBILITY

To ensure the reproducibility of our results, we will release in mid-July software summarized in Tab. 3. Our Linux kernel code was based on the most recent Multipath TCP release (v0.91) that was backported to kernel v3.4 to be usable on the Nexus 5 smartphones. We will also release all the Mininet simulations, the modified Android 6.0.1 ROM and the measurement applications we developed. A companion website will provide the anonymized Multipath TCP packet traces.

REFERENCES

[1] 2010. Cisco Visual Networking Index. (February 2010). http://www.cisco.com/c/en/us/solutions/collateral/ service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html.

[2] 2015. Improve active/backup subflow selection. (January 2015). https://github.com/multipath-tcp/mptcp/pull/70.

[3] 2016. Android Open Source Project source code. (Dec. 2016). Branch android-6.0.1-r77 of https://android.googlesource.com/platform/frameworks/base.

[4] A. Asheralieva, T. J. Erke, and K. Kilkki. 2009. Traffic characterization and service performance in FON network. In *Future Information Networks, 2009. ICFIN 2009. First International Conference on*. IEEE, 285–291.

[5] M. Assefi, G. Liu, M. P. Wittit, and C. Izurieta. 2016. Measuring the Impact of Network Performance on Cloud-Based Speech Recognition Applications. *International Journal of Computer Applications-IJCA* 23 (2016), 19–28.

[6] M. Bagnulo. 2011. Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6181. (March 2011). https://rfc-editor.org/rfc/rfc6181.txt

[7] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. 2009. Energy consumption in mobile phones: a measurement study and implications for network applications. In *IMC'09*. ACM, 280–293.

[8] O. Bonaventure and S. Seo. 2016. Multipath TCP Deployments. In *IETF Journal*. Vol. 12. Internet Society, 24–27.

[9] B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, J. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl. 2014. Reducing internet latency: A survey of techniques and their merits. *IEEE Communications Surveys & Tutorials* 18, 3 (2014), 2149–2196.

[10] A. Carroll and G. Heiser. 2010. An Analysis of Power Consumption in a Smartphone.. In *USENIX annual technical conference*, Vol. 14. Boston, MA.

[11] X. Chen, A. Jindal, N. Ding, Y. C. Hu, M. Gupta, and R. Vannithamby. 2015. Smartphone background activities in the wild: Origin, energy drain, and optimization. In *Mobicom'15*. ACM, 40–52.

[12] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley. 2013. A measurement-based study of multipath tcp performance over wireless networks. In *IMC'13*. ACM, 455–468.

[13] Y. Cheng, J. Chu, A. Jain, and S. Radhakrishnan. 2014. TCP Fast Open. RFC 7413. (Dec. 2014). https://doi.org/10.17487/rfc7413

[14] S. Cheshire, D. Schinazi, and C. Paasch. 2017. Advances in Networking, Part 1. (June 2017). https://developer.apple.com/videos/play/wwdc2017/707//.

[15] Q. De Coninck, M. Baerts, B. Hesmans, and O. Bonaventure. 2016. A First Analysis of Multipath TCP on Smartphones. In *PAM'16*. Springer, 57–69.

[16] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan. 2014. Wifi, lte, or both?: Measuring multi-homed wireless internet performance. In *IMC'14*. ACM, 181–194.

[17] N. Dukkipati, N. Cardwell, Y. Cheng, and M. Mathis. 2013. Tail Loss Probe (TLP): An algorithm for fast recovery of tail losses. *IETF Draft, draft-dukkipati-tcpm-tcploss-probe-01* (2013).

[18] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. 2010. A First Look at Traffic on Smartphones. In *IMC '10*. ACM, New York, NY, USA, 281–287. https://doi.org/10.1145/1879141.1879176

[19] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. 2010. Diversity in smartphone usage. In *MobiSys'10*. ACM, 179–194.

[20] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. 2013. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824. (January 2013).

[21] A. Frommgen, T. Erbshäußer, A. Buchmann, T. Zimmermann, and K. Wehrle. 2016. Remp TCP: Low latency multipath TCP. In *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 1–7.

[22] B. Han, F. Qian, S. Hao, and L. Ji. 2015. An Anatomy of Mobile Web Performance over Multipath TCP. In *CoNEXT '15*. ACM, New York, NY, USA, Article 5, 7 pages.

[23] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. 2012. Reproducible network experiments using container-based emulation. In *CONEXT'12*. ACM, 253–264.

[24] S. He, Y. Liu, and H. Zhou. 2015. Optimizing smartphone power consumption through dynamic resolution scaling. In *Mobicom'15*. ACM, 27–39.

[25] J. Hempel. 2016. Voice Is the Next Big Platform, and Alexa Will Own It. (Dec. 2016). https://backchannel.com/voice-is-the-next-big-platform-and-alexa-will-own-it-c2cf13fab911#.3eewnvjbp.

[26] B. Hesmans and O. Bonaventure. 2016. An enhanced socket API for Multipath TCP. In *ANRW'16*. ACM, 1–6.

[27] X. Hu, L. Song, D. Van Bruggen, and A. Striegel. 2015. Is there wifi yet? how aggressive wifi probe requests deteriorate energy and throughput. *arXiv preprint arXiv:1502.01222* (2015).

[28] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. 2012. A close examination of performance and power characteristics of 4G LTE networks. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 225–238.

[29] W. Knight. Conversational Interfaces. *MIT Technology Review* (????).

[30] M. Li, A. Lukyanenko, Z. Ou, A. Yla-Jaaski, S. Tarkoma, M. Coudron, and S. Secci. 2016. Multipath transmission for the internet: A survey. *IEEE Communications Surveys Tutorials, vol. PP* 99 (2016), 1–41.

[31] Y.-s. Lim, Y.-C. Chen, E. M. Nahum, D. Towsley, and R. J. Gibbens. 2014. How green is multipath TCP for mobile devices?. In *Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges*. ACM, 3–8.

[32] Y.-s. Lim, Y.-C. Chen, E. M. Nahum, D. Towsley, R. J. Gibbens, and E. Cecchet. 2015. Design, implementation, and evaluation of energy-aware multi-path TCP. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. ACM, 30.

[33] A. Nika, Y. Zhu, N. Ding, A. Jindal, Y. C. Hu, X. Zhou, B. Y. Zhao, and H. Zheng. 2015. Energy and performance of smartphone radio bundling in outdoor environments. In *Proceedings of the 24th International Conference on World Wide Web*. ACM, 809–819.

[34] A. Nikravesh, Y. Guo, F. Qian, Z. M. Mao, and S. Sen. 2016. An in-depth understanding of multipath TCP on mobile devices: measurement and system design. In *Mobicom'16*. ACM, 189–201.

[35] B.-H. Oh and J. Lee. 2015. Constraint-based proactive scheduling for MPTCP in wireless networks. *Computer Networks* 91 (2015), 548–563.

[36] C. Paasch, S. Barre, et al. 2017. Multipath TCP in the Linux Kernel. (2017). http://www.multipath-tcp.org.

[37] C. Paasch, G. Detal, F. Duchene, C. Raiciu, and O. Bonaventure. 2012. Exploring mobile/WiFi handover with Multipath TCP. In *CellNet'12*. ACM, 31–36.

[38] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure. 2014. Experimental evaluation of multipath TCP schedulers. In *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*. ACM, 27–32.

[39] C. Paasch, R. Khalili, and O. Bonaventure. 2013. On the benefits of applying experimental design to improve multipath TCP. In *CONEXT'13*. ACM, 393–398.

[40] C. Pluntke, L. Eggert, and N. Kiukkonen. 2011. Saving mobile device energy with multipath TCP. In *Proceedings of the sixth international workshop on MobiArch*. ACM, 1–6.

[41] M.-R. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, and M. J. Neely. 2010. Energy-delay tradeoffs in smartphone applications. In

Proceedings of the 8th international conference on Mobile systems, applications, and services. ACM, 255–270.

[42] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan. 2011. TCP Fast Open. In *CONEXT'11*. ACM, 21.

[43] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. 2011. Improving datacenter performance and robustness with multipath TCP. In *ACM SIGCOMM Computer Communication Review*, Vol. 41. ACM, 266–277.

[44] C. Raiciu, D. Niculescu, M. Bagnulo, and M. J. Handley. 2011. Opportunistic mobility with multipath TCP. In *Proceedings of the sixth international workshop on MobiArch*. ACM, 7–12.

[45] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. 2012. How hard can it be? designing and implementing a deployable multipath TCP. In *NSDI'12*. USENIX Association, 29–29.

[46] E. Rescorla. 2017. The Transport Layer Security (TLS) Protocol Version 1.3. (April 2017). Internet draft, draft-ietf-tls-tls13-20.

[47] S. Seo. 2015. KTâĂŹs GiGA LTE. Presented as the 93th IETF in Prague, Czech Republic. (2015). https://www.ietf.org/proceedings/93/slides/slides-93-mptcp-3.pdf

[48] H. Sinky, B. Hamdaoui, and M. Guizani. 2016. Proactive Multipath TCP for Seamless Handoff in Heterogeneous Wireless Access Networks. *IEEE Transactions on Wireless Communications* 15, 7 (2016), 4754–4764.

[49] J. Sommers and P. Barford. 2012. Cell vs. WiFi: on the performance of metro area mobile connections. In *IMC'12*. ACM, 301–314.

[50] N. Vallina-Rodriguez and J. Crowcroft. 2011. ErdOS: achieving energy savings in mobile OS. In *MobiArch'11*. ACM, 37–42.

[51] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. 2012. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring. In *USENIX Annual Technical Conference (USENIX ATC 12)*. 387–400.

[52] Y. Zhu, Y. Zhu, A. Nika, B. Y. Zhao, and H. Zheng. 2016. Trimming the Smartphone Network Stack. In *HotNets '16*. ACM, New York, NY, USA, 176–182.