An In-depth Understanding of Multipath TCP on Mobile Devices: Measurement and System Design

Ashkan Nikravesh*
University of Michigan
Ann Arbor, MI
ashnik@umich.edu

Yihua Guo*
University of Michigan
Ann Arbor, MI
yhguo@umich.edu

Feng Qian
Indiana University
Bloomington, IN
fengqian@indiana.edu

Z. Morley Mao
University of Michigan
Ann Arbor, MI
zmao@umich.edu

Subhabrata Sen
AT&T Labs – Research
Bedminster, NJ
sen@research.att.com

ABSTRACT

Today's mobile devices are usually equipped with multiple wireless network interfaces that provide new opportunities for improving application performance. In this paper, we conduct an in-depth study of multipath for mobile settings, focusing on MPTCP, with the goal of developing key insights for evolving the mobile multipath design. First, we conduct to our knowledge the most in-depth and the longest user trial of mobile multipath that focuses not only on MPTCP performance, but also on cross-layer interactions. Second, we identify a new research problem of multipath-aware CDN server selection. We demonstrate its real-world importance and provide recommendations. Third, our measurement findings lead us to design and implement a flexible software architecture for mobile multipath called MPFlex, which strategically employs multiplexing to improve multipath performance (by up to 63% for short-lived flows). MPFlex decouples the high-level scheduling algorithm and the low-level OS protocol implementation, and enables developers to flexibly plug-in new multipath features. MPFlex also provides an ideal vantage point for flexibly realizing user-specified multipath policies and is friendly to middleboxes.

CCS Concepts

•**Networks** → **Transport protocols; Network measurement;**

1. INTRODUCTION

The support for multiple network interfaces is a norm on today's smart devices: smartphones and tablets often have both WiFi and cellular connectivity; wearable devices are capable of pairing with their phones using either Bluetooth or Direct WiFi; even Internet-of-Things (IoT) devices such as home alarms and smoke detectors can potentially leverage multipath for traffic offloading [40].

Multipath provides new opportunities for improving mobile application performance. The most widely used multipath solution is Multipath TCP (MPTCP) [18], which allows unmodified applications to transfer data over multipath. In the research community, numerous studies have been conducted on multipath [13, 17, 26, 25, 14, 28, 19, 39, 20]. Industry has also been enthusiastically adopting multipath. Some built-in apps in iOS, *e.g.,* Siri, support multipath [2]. A Korean R&D Center plans to commercialize "GiGA LTE", which achieves Gbps throughput on commodity smartphones over multipath [3].

Despite these efforts, there still remain numerous challenges for effectively and efficiently using mobile multipath. To name a few, first, originally designed for data center networking [34], MPTCP may incur unexpected cross-layer interactions when used on mobile devices [19, 17]. Second, as shown later, MPTCP often incurs additional energy overhead, but may not always boost (sometimes even worsen) application performance. Therefore, applications should use multipath only when its benefit outweighs its incurred overhead. Such decision logic is largely missing or done naïvely in practice. Third, protocols such as MPTCP are complex with numerous configurations, and it is unclear how to tune them in an optimal way. Fourth, from a system architectural perspective, MPTCP's "everything in kernel" scheme may not be suitable for mobile multipath to support a rich set of application-specific policies.

In this paper, we conduct an in-depth measurement-driven study of multipath over mobile devices, with the goal of providing key knowledge and vital clues for evolving the mobile multipath design. More specifically, we explore the following unanswered questions.

• *How much performance benefits can MPTCP offer in real-world settings?* Differing from prior studies [13, 17, 26, 19] performing in-lab controlled experiments, we launch an IRB-approved user trial[1] by collecting network traces from real users' smartphones with MPTCP enabled. Meanwhile, we complement the passive measurements with crowd-sourced active probing on participants' phones to capture application quality of experience (QoE) over multipath under diverse network conditions. To our knowledge, this is the most in-depth user study of mobile multipath that focuses not only on multipath itself, but also on cross-layer interactions. Leveraging the unique data we collected, we analyzed MPTCP behaviors "in the wild", including multipath

*Co-primary authors.

MobiCom'16, October 03-07, 2016, New York City, NY, USA

DOI: http://dx.doi.org/10.1145/2973750.2973769

[1]This study was conducted entirely with data collected from active and passive measurements at the University of Michigan and was approved by the University of Michigan IRB (Institutional Review Board) approval number HUM00111075.

availability, path utilization, handshake latency, TCP throughput, web page load time, video streaming bitrate. We also examined multipath for voice-over-IP (VoIP) and instance messenger (IM) whose multipath-friendliness have not been explored before. From the user study, we found that multipath is widely available and it incurs rather complex interactions with applications due to their diverse traffic patterns and QoE metrics. MPTCP's suboptimal performance often stems from three factors: short flow duration, excessive delay of subflows' handshakes, and the scheduling algorithms.

• *What is the energy footprint of MPTCP on real users?* MPTCP incurs additional energy overhead [39, 26]. We analyzed this overhead by applying previously validated single-path and multipath radio energy models on real users' network traffic. We found that properly using multipath incurs reasonable and manageable overhead. However, blindly applying MPTCP to all traffic, as is usually done today, increases users' average radio energy by 1.08X. Our study shows that many optimizations can be performed to reduce multipath's energy footprint, such as improving the scheduling algorithm for small transfers and disabling multipath for applications such as VoIP and IM.

• *How to improve the interplay between multipath and server selection?* We consider a classic problem of CDN server selection. We found that under multipath, the state-of-the-art DNS-based server selection often leads to suboptimal server selection. This is attributed to the fact that CDNs' DNS infrastructure is unaware of other subflows available to the client. Compared to single-path, under multipath, choosing different CDN servers incurs more complex tradeoffs, and can lead to different network performance depending on MPTCP's scheduling algorithm, traffic patterns, and path characteristics. We are the first to identify the problem of CDN server selection under multipath. We demonstrate its real-world importance by probing Alexa top-500 websites and provide recommendations for multipath-aware server selection.

• *How to improve the system architecture for mobile multipath?* The measurement results indicate that MPTCP suffers from a few limitations: poor interaction with short/small flows, a lack of infrastructural support for multipath policy, and MPTCP extension often being blocked by middleboxes. We propose a flexible software architecture of mobile multipath called MPFlex that overcomes all the above limitations. MPFlex has several prominent features. First, it performs transparent *multiplexing* for application traffic over multipath. Our multiplexing scheme reduces the number of handshakes from many (one per path) to zero, leading to significant improvement of bandwidth utilization for small flows. Second, MPFlex *decouples* the high-level scheduling algorithm and the low-level OS protocol implementation. This is realized by maintaining most of MPFlex's logic in the user space, which obtains lower-layer information (*e.g.,* latency and congestion window) from kernel through a unified API. Such a framework dramatically simplifies the development, deployment, and maintenance of multipath features. Third, MPFlex has visibility of all traffic on an end host, and thus provides an ideal vantage point for applying user-specified multipath policies. Fourth, MPFlex is middlebox-friendly as it does not use any Layer 3 or 4 protocol extensions which may be blocked by ISPs. Compared to MPTCP, MPFlex reduces single file transfer time by up to 49%, improves bundled short flows' transfer time by up to 63%, and boosts real web page load speed by up to 20%, while incurring negligible overhead. We also demonstrate MPFlex's capability of flexibly plugging-in new features such as buffer-aware scheduling, smart packet reinjection, and per-application policies, which can be implemented in less than 70 lines of user-level code.

Overall, we make key contributions to mobile multipath research in two aspects: crowd-sourced measurement (§2 and §3) and improved software architecture (§4 and §5). We discuss related work in §6 before concluding the paper in §7.

2. MEASUREMENT METHODOLOGY

We describe our IRB-approved user trial to reveal characteristics of mobile multipath traffic "in the wild".

2.1 Multipath Configuration For User Trial

We consider a common usage scenario where WiFi and cellular are used at the same time on a smartphone. To realize this, we used MPTCP, the most popular multipath solution with off-the-shelf Linux kernel implementation [27]. We port MPTCP v0.86 to Android 4.4.4 with CyanogenMod 11 ROM on Samsung Galaxy S3 (SGS3) smartphones. MPTCP enables all applications to transparently utilize multipath. However, one challenge is most today's servers do not yet support MPTCP. We thus set up a multipath proxy running MPTCP v0.90 on a server with 64-core 2.6GHz CPU, 128GB memory, and 64-bit Ubuntu 14.04 installed. To redirect traffic to the MPTCP proxy, we transparently tunnel all user traffic using Socks5 proxying (using `shadowsocks` [6]). The Socks5 protocol [24] only adds a very small header to each packet so its impact on the traffic pattern is negligible. We also verified Socks5 incurs very small runtime overhead.

We recruited 15 students studying at the University of Michigan and gave each a SGS3 smartphone with unlimited cellular data plan of a commercial U.S. cellular carrier. The participants were asked to use the phone normally with WiFi and cellular enabled. We expect multipath is available at least on campus, at participants' home, and in places with free public WiFi.

The proxy is connected to the University of Michigan campus network. This makes the latency of the WiFi path small when participants use the phones on campus. In our data, only 14% of the traffic belong to this case. We separate out such traffic in some of our analysis. We also systematically study the impact of the proxy location in §5.5.

2.2 User Trial Data Collection

We built a custom data collector running on users' devices. It transparently performs three tasks: passive measurements, active measurements, and data upload. The collector incurs small overhead without noticeable degradation of users' experience, based on our lab testing.

Passive Measurements. The data collector passively collects network packet traces (only TCP/IP headers) via a modified `tcpdump` on users' phones (we did not capture traces at the proxy because for some analysis such as energy, we must use client-side traces). Also, the collector re-configures MPTCP settings of its device every 24 hours by randomly selecting one of the three configurations: (1) MPTCP is disabled, (2) MPTCP is enabled with WiFi selected as the primary path, and (3) MPTCP is enabled with cellular as the primary path. Doing so allows us to statistically compare MPTCP and SPTCP (single-path TCP), as well as to study the impact of the primary path selection.

Active Measurements. To complement passive measurements, the collector periodically (every hour) runs the following active measurements back-to-back in background: (1) stream a 2-min YouTube video; (2) download a file over single- and multipath; (3) load five popular webpages using different configurations of SPTCP and MPTCP. We detail the measurement methodologies in §3.2. The collector records key performance metrics such as

Table 1: Summary of the findings and proposed improvement

Measurement Findings	Proposed Improvement
Inefficiencies for small flows (§3.1): (1) low path utilization for the secondary path, (2) low MPTCP throughput, (3) excessive energy consumption.	Multiplexing and 0-RTT multipath connection establishment (§4.2). Flexible scheduling algorithm design (§4.3).
Performance and resource impact of MPTCP on different applications differs. (§3.1, §3.2, and §3.3)	A framework supporting multipath policy (§4.2).
SPTCP may outperform MPTCP when network conditions of two paths differ, caused by receiver-side out-of-order (§3.2).	Dynamic and configurable packet reinjection (§4.3).
Sub-optimal CDN server selection for MPTCP connections (§3.4).	Multipath-aware CDN selection (§3.4).

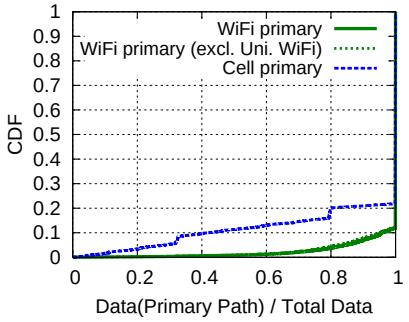


Figure 1: Distributions of the fractions of payload transmitted over the primary subflow, across all MPTCP flows.

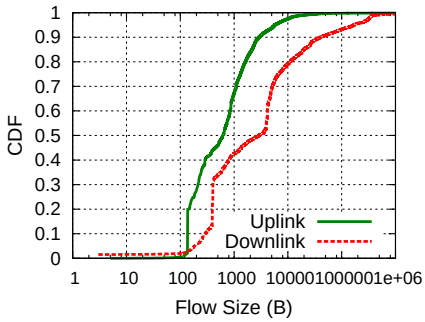


Figure 2: Distributions of DL/UL bytes in a SPTCP/MPTCP flow.

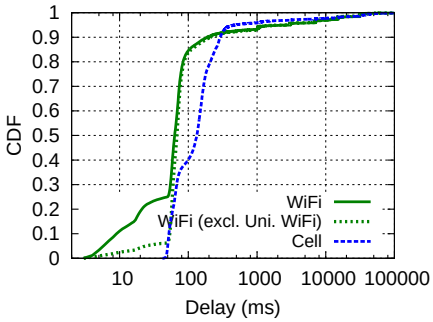


Figure 3: Distributions of the handshake delays of MPTCP secondary subflow.

Table 2: Statistics of the user study dataset.

Passive Measurements			Active Measurements		
Apps	All TCP Connections*	MPTCP Conns.	Page loads	Video plays	File DLs
122	1516794	441422	24668	3071	4371

* We use "connection" or "flow" interchangeably when referring to a SPTCP/MPTCP connection, and use "subflow" for MPTCP subflow.

video QoE (*i.e.,* video quality and rebuffering events) and page load time, which are difficult to obtain from passive measurements. Note the active measurements are only conducted when the screen is off, the battery life permits, and the phone is idle, so it does not impact user experience.

Data Upload. The collected data for both passive and active measurements is uploaded to our measurement server over every day at late night when the phone is charging. We collected the data for more than 4 months (33 weeks) from October 25, 2015 from the 15 participants, with general statistics shown in Table 2.

Controlled In-lab Experiments. Besides the user trial, we also conduct in-lab experiments, which serve two purposes: validating our findings/systems, and covering experiments that are too difficult to conduct in the user trial. Unless otherwise specified, we used a Nexus 5 phone with Android 4.4.4 and MPTCP v0.89.5 for our in-lab experiments conducted in §3.3 and §3.4.

3. MEASUREMENT RESULTS

We now present our measurement results summarized in Table 1 for both user study and controlled measurement.

3.1 Passive Measurements of User Study

Multipath availability. The first question to answer is, how often is multipath available? During the data collection period, for 82% and 40% of the time, users have WiFi and cellular connectivity, respectively. Given cellular (WiFi) path is available, for 73% (42%) of the time, users can leverage both paths. Overall, the results indicate that multipath is common on mobile devices.

Path Utilization. When multipath is available, how effectively are the two paths utilized? Figure 1 plots the distributions of the fraction of payload (both uplink and downlink) transmitted over the primary subflow, across all MPTCP connections. When WiFi is used as the primary path, more than 89% of the flows are fully delivered over WiFi. This is attributed to two reasons. First, the vast majority of the flows have small sizes. As shown in Figure 2, the 75% percentiles of uplink and downlink bytes within a TCP flow (single- or multipath) are only 1.3KB and 6.7KB, respectively. Second, MPTCP by default performs subflow handshakes sequentially *i.e.,* handshake of the secondary subflow occurs after the completion of the handshake of the primary subflow. As a result, for short WiFi-primary flows, often the user data can be fully delivered over WiFi before the LTE subflow is established.

Figure 3 plots distributions of the handshake delay of the secondary subflows. The median handshake delays of cellular and WiFi secondary subflows are 133ms and 63ms, respectively. As a result, when LTE becomes the primary path, because the WiFi path is established quicker (compared to the LTE path establishment when WiFi is the primary path), there are more opportunities for WiFi to be utilized, as shown in Figure 1. Excluding flows whose primary paths connect to the university WiFi (they account for only 11% of the total flows) slightly shifts the low-end of the distribution.

On the other hand, as the flow size increases, the fraction of primary-path bytes decreases. This is because large flows' longer duration allows both paths to be established and transfer non-trivial amount of data. For all MPTCP flows, 69% of the bytes are transferred over WiFi and 31% are over cellular. A large fraction of these bytes are contributed by a small fraction of large flows due to the heavy-tail distribution of flow sizes [30].

Also note MPTCP can be configured to perform both handshakes simultaneously. This can improve the overall throughput due to shorter handshake time, but it incurs energy overhead, and may bring limited improvement when one path has long latency. We propose a scheme in §4 that boosts the multipath performance by reducing the number of handshakes from many to zero.

MPTCP Performance. How much performance benefits does MPTCP provide? Here we focus on a simple metric – the transport-layer throughput, and study application QoE later in §3.2 and §3.3. Figure 4 studies five flow size groups within each four schemes are compared: SPTCP over cellular, SPTCP over WiFi, cellular-primary MPTCP, and WiFi-primary MPTCP. We normalize the measured throughput at a per-group basis with the maximum throughput in each group (normalized to 1) annotated. Also note for Figure 4, we ignore all flows with inter-packet-arrival time greater than 1s to ensure the flow is not likely to be throttled by application.

We make two observations from Figure 4. First, for small flows (less than 100KB), SPTCP over WiFi provides good throughput that usually outperforms MPTCP. This is because latency plays a more important role in determining small flows' performance than bandwidth does, and in our dataset WiFi usually has a smaller RTT than cellular so transferring the entire flow over WiFi may provide better performance. A question one may ask is, given MPTCP's scheduler selection the path with the smallest RTT, why does MPTCP not perform at least as well as SPTCP? The reason is, the above path selection only happens when both paths have spaces in congestion window (cwnd). If, for example, WiFi's cwnd is full but LTE has empty cwnd space, the data will still be transferred over LTE even if its latency is higher. This explains why MPTCP may incur worse performance than SPTCP does. We will improve this in §4.

The second observation from Figure 4 relates to large file download where throughput is more important than latency. In this case MPTCP always achieves higher throughput than SPTCP does. Here we see the selection of the primary path still matters even for large files. WiFi-primary MPTCP outperforms cellular-primary because, as mentioned before, the long subflow handshake time of LTE prolongs the overall download time. Also note that for both MPTCP and SPTCP, flows with larger sizes achieve statistically higher throughput, as the links are more likely to be saturated for larger flows.

The performance of both SPTCP and MPTCP is also affected by the latency of the corresponding path(s). In our dataset, WiFi usually has a smaller RTT than cellular (even when we exclude the university WiFi traffic that naturally has low WiFi RTT between the client and the proxy). This aligns with recent measurement studies of WiFi and cellular performance [17, 36]. We will systematically study the performance impact of proxy location (and thus the latecny) in §5.5.

Radio Energy is the energy consumed by the radio interface. It accounts for a significant fraction of the overall mobile device energy consumption in particular for cellular (1/3 to 1/2 for 3G [32] and at least 50% for LTE [26]). Multipath impacts the radio energy consumption in two ways. First, in many cases, it obviously incurs additional energy footprint by using multiple interfaces. Second, if properly leveraged, it may reduce the energy consumption by shortening the file transfer time. In reality, how energy (in)efficient is MPTCP compared to SPTCP? To answer this, we feed the collected trace using the LTE/WiFi radio energy models developed by Nika *et al.* [26] for SGS3, and compute the radio energy consumption. Recall that in the user study, for each user, the data collector re-configures its multipath setting every 24 hours. We found that for MPTCP configurations, their average radio power is 2.08 times of that for the SPTCP configuration. Based on this, we can roughly estimate the impact of multipath on the overall device battery drain. Assuming for single-path, the radio power (either WiFi or cellular) accounts for 25% of the overall device energy consumption [12], enabling *system-wide* MPTCP shortens the battery life by 21%. We admit though this is a coarse-grained estimation, and it is an upper bound that does not consider the reduced radio energy due to MPTCP's shorter transfer time. We believe this overhead is not unreasonably high, and expect it to be further reduced by using energy-aware policies described later.

Mobile Apps over MPTCP. We pick the five most-heavily used apps by the 15 users over MPTCP to study the resource impact of the cellular subflow. For each app, we plot the fraction of cellular bytes and cellular radio energy as shown in Figure 5. For four out of the five apps, less than 10% of the bytes are delivered over cellular (due to the small flow sizes as explained earlier), while the cellular energy accounts for more than 50% of the total radio energy consumption. For most small flows using WiFi-primary MPTCP, additional energy penalty comes from the fact that even if LTE carries no user payload, the LTE interface still needs to be activated for handshake. This occurs in 91% of the flows with sizes less than 100KB. For these flows, 63% of the radio energy is consumed by cellular that does not deliver any user payload. The energy efficiency increases with larger flows, *e.g.,* for YouTube.

Overall, our findings suggest that although multipath is energy-wise expensive, by strategically leveraging it in an energy-aware manner, the energy overhead can be reasonable and manageable. Some recent work such as eMPTCP [39] has made a first step toward this goal. We believe that besides improving the scheduler [39], another important premise for energy-aware multipath is to have per-application policy, as different apps incur different cost-benefit tradeoffs when using multipath. We study this in more depth in §3.2 and §3.3, and propose a system to allow easy deployment of user policies (§4).

3.2 Active Measurements of User Study

We now shift our focus to the crowd-sourced active measurement results. Complementing the passive measurement, active measurements provide insights of how multipath impacts the application performance.

File download. We collected 4,371 measurements of downloading a 10MB file using single- and multipath. MPTCP improves the file download performance by 34% in average. Note that the MPTCP throughput is smaller than the sum of both paths' throughput. We found this is because MPTCP's short flow duration makes the congestion window not fully expanded, compared to SPTCP flows.

We next study the MPTCP performance under diverse wireless link qualities. Figure 6 plots the ratio between throughput of MPTCP and SPTCP (WiFi-primary), whose measurements were conducted back-to-back, under three ranges of LTE signal strength. Ideally, MPTCP should be at least as good as SPTCP. Surprisingly, when LTE signal strength is poor, for 20% of downloads, SPTCP provides higher throughput than MPTCP. We found this is due to the limited receive window size. Compared to SPTCP, MPTCP requires a larger (connection-level) receive window buffer to absorb out-of-order arrival from multiple paths in particular when the paths have diverse qualities. The default upper limit of the receive window is set too small on Android. We confirmed this through controlled experiments (good WiFi quality, -116dbm LTE signal strength), and observed a strong correlation (Pearson coefficient of 0.79) between MPTCP instantaneous throughput

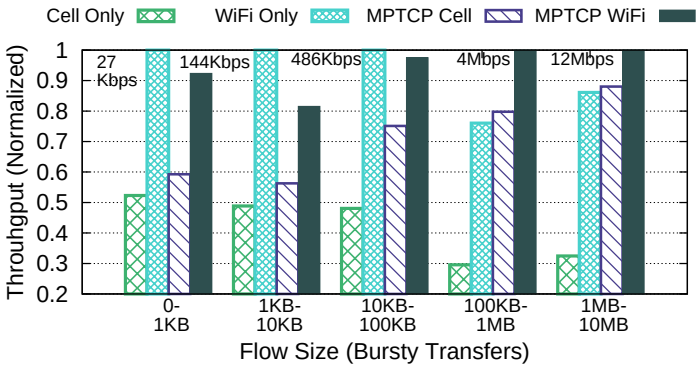


Figure 4: Average TCP throughput for different flow size groups.

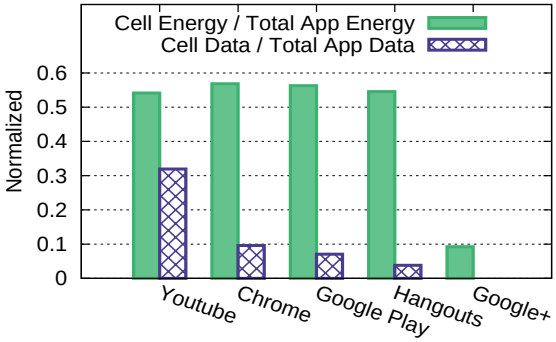


Figure 5: The fraction of data delivered by cellular versus the fraction of energy consumed by cellular for popular apps.

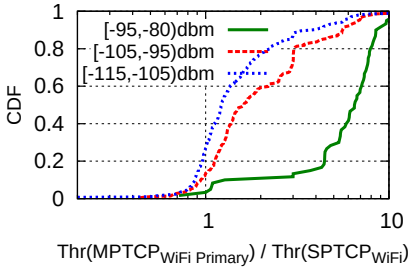


Figure 6: Throughput ratio of MPTCP (WiFi primary) to SPTCP over WiFi for 4,371 back-to-back throughput measurements for different LTE signal strength levels.

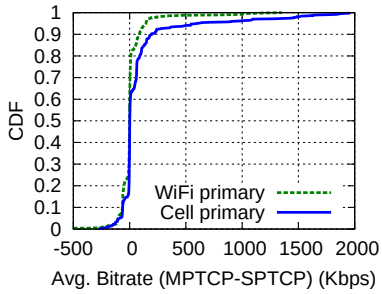


Figure 7: Crowd-sourced video streaming measurements (1,500 measurements in total).

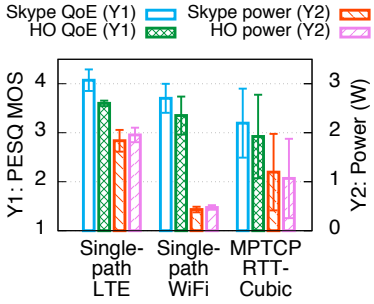


Figure 8: QoE and power consumption of VoIP under different SPTCP/MPTCP settings.

and available receive window space, indicating that decrease in available receiver window causes the sender to slow down the rate on both paths. When we increase the upper limit of the receive window from the default value (1MB) by 1% (10KB), we observe 12% reduction in download time.

Web browsing. We obtained 24K measurements of web page loadings, across five popular sites. We found MPTCP provides small improvement of page load time compared to the best SPTCP result (improvement ranging from 1% to 7%). Inline with prior in-lab measurement on laptops [19], it is attributed to HTTP's traffic pattern of multiple short-lived connections that leave small chances for the secondary subflow to be used. This problem can potentially be mitigated by HTTP/2 that uses multiplexing. But using an HTTP/2 proxy incurs other limitations as to be discussed in §4 where we describe our transport-layer multiplexing proposal over multipath.

Video Streaming. We conducted about 3,000 crowd-sourced measurements of capacity-based adaptive (CBA) video streaming, by playing a 2:15 YouTube video with four available bitrates: 240p, 360p, 480p, and 720p. The CBA [35] selects the bitrate based on the current capacity (*i.e.,* download time of recently fetched chunks). Figure 7 measures the difference of the average bitrate between MPTCP and SPTCP. For about 55% of all measurements, MPTCP provides no improvement of the video quality, because either the whole video is already being played at the highest bitrate, or more often (64%), the additional throughput provided by MPTCP is not enough for switching to a higher bitrate. Nevertheless, we do observe less frequent rebuffering events when MPTCP is used, compared to SPTCP: the average number of rebuffering is 0.24, 0.29, 0.21, and 0.18 for SPTCP over WiFi, SPTCP over cellular, WiFi-primary MPTCP, and cellular-primary MPTCP, respectively. This implies MPTCP can provide more robustness for video streaming.

3.3 Other Applications over MPTCP: Voice-over-IP and Instant Messengers

We now study two other important applications (voice-over-IP and instant messengers) in controlled experiments due to the difficulty of capturing their QoE in the user study.

Voice-over-IP (VoIP) is a representative real-time application requiring low latency and small jitter (variation of latency). We study two popular VoIP apps: Skype and Google Hangouts using the University of Michigan campus WiFi and a commercial LTE carrier. The WiFi has smaller RTT than LTE[2] (20ms vs. 60ms) between two clients in our lab: an Ethernet-connected desktop and a Google Nexus 5 smartphone. For both applications, we play a 2-min pre-recorded audio from the desktop client for 10 times and record the received audio at the phone side to compute the QoE using PESQ MOS [4], which compares the original and the received VoIP audio. PESQ MOS is a number between 1.0 and 4.5, with 4.5 representing the best quality. We block UDP to force the app to use TCP, and will consider UDP shortly.

Figure 8 (Y1 Axis) plots the VoIP QoE using SPTCP (over LTE and WiFi) and WiFi-primary MPTCP with different schedulers (RTT for minimum-RTT and RR for round robin). We found

[2]A recent measurement study [17] reports that WiFi latency outperforms LTE 80% of times and the median latency difference between WiFi and LTE is 30-40ms.

that compared to SPTCP, MPTCP with the min-RTT scheduler actually worsens the VoIP QoE. This is due to the increased latency variation, from 24ms for LTE and 12ms for WiFi to 97ms for MPTCP, and additional receiver-side buffering delay of out-of-order packets, from 8ms for SPTCP to 82ms for MPTCP. Both factors contribute to the significant increase of *application-observed* jitter that lowers the PESQ score. We also increased the LTE latency to 100ms and observed qualitatively similar results. Overall, our findings suggest that real-time applications requiring low jitter may not benefit from MPTCP, in particular when the two paths have diverse latency. To validate this under UDP, we write a custom program that sends a 100-byte UDP datagram every 10ms over single path or multipath with round robin, and measure the jitter and packet out-of-order at the receiver side. We observed when multipath is used, the variation of UDP one-way delay increases by up to 50% compared to SPTCP, and about 50% of the UDP datagrams are delivered out-of-order. Note although UDP itself does not buffer out-of-order datagrams, the application may either buffer or drop them, both resulting in degraded QoE.

Meanwhile, we employ the model used in §3.1 to compute the radio energy consumption for each scheme in Figure 8. As shown (Y2 Axis), compared to SPTCP over WiFi, MPTCP incurs additional radio power of 125% to 177% for VoIP, making it even more undesirable to use MPTCP.

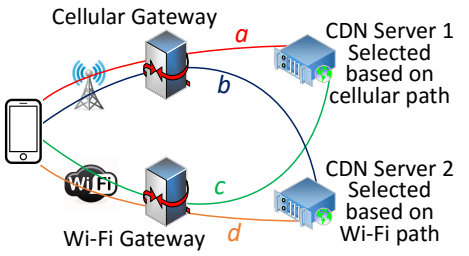
Instance Messenger (IM). We study three popular IM apps: Facebook, Google Hangout, and Whatsapp. We test them by sending a message every 30 seconds from one phone to another over SPTCP (WiFi only) and WiFi-primary MPTCP with the min-RTT scheduler. We measure the message delivery delay, the key QoE metric, at sender. We also compute the radio energy consumption. We found MPTCP increases the message delivery delay by 5% to 36% compared to SPTCP. This is due to the transmission of small data on the higher-latency LTE subflow (explained in §3.1). MPTCP incurs additional radio energy consumption of 18% to 72% for the message delivery. Overall, we found using MPTCP for IM incurs both performance and energy penalty, and thus should be avoided.

Middlebox Friendliness. We also investigate whether the TCP option used by MPTCP can pass middleboxes that are prevalent in cellular networks [44, 21]. We found that middleboxes of at least two commercial cellular carriers in the U.S. strip TCP options from TCP headers. This motivates multipath solutions that do not require any TCP/IP extensions.

3.4 Interplay between Multipath and CDN

In this section, we consider a classic problem of server selection in the context of multipath. Due to the wide use of Content Delivery Network (CDN), the same content (*e.g.,* a web page or a video chunk) is often replicated across servers at multiple geographically distributed locations. A user request will be directed to a CDN location, which is usually close to the client, that provides the best user experience. Most CDNs achieve this by leveraging local DNS (LDNS) server with CDN server selection algorithms based on information such as geo-location databases [8].

Measurement for CDN over Multipath. DNS-based server selection can still work in multipath. But here we have a new issue that has not been explored by the literature to our knowledge. Since the DNS request is sent by only one path's (usually the default path's) LDNS server, the CDN is not aware of the existence of other paths. Therefore, although the selected server provides good performance for one path, it may be sub-optimal for other paths and therefore for MPTCP. We conducted an emulated experiment to illustrate this. In Figure 9, let Server 1 and 2 be the optimal



Path	*a*	*b*	*c*	*d*
BW (Mbps)	6.0	6.0	3.0	3.0
RTT (ms)	80	80 + *x*	60 + *x*	60

Figure 9: Example: CDN sever selection over multipath.

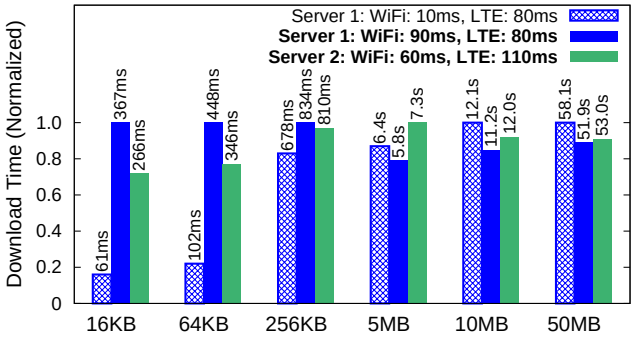


Figure 10: File download time from different CDN servers, averaged over 10 runs. Shaded blue is a "what-if" scenario.

CDN servers selected based on the cellular and the WiFi path, respectively. If we use Server 1 (Server 2), then MPTCP will send traffic over path a and c (b and d) for cellular and WiFi, respectively. Figure 9 also lists the characteristics of the four paths. We assume that a and b (also c and d) have the same bandwidth since they share the same "last-mile" that is usually the bandwidth bottleneck. But when WiFi (cellular) accesses the server selected by cellular (WiFi), a latency penalty x=30ms caused by additional queuing and propagation delay will be added (on path b and c).

We use MPTCP with the min-RTT scheduler to download files with various sizes from both servers. MPTCP uses path a and c for Server 1, and b and d for Server 2. We discuss two scenarios: downloading small files and large files.

Download small files. As shown on the left side of Figure 10, when the files are small, CDN Server 2 gives shorter download time than Server 1 does. This is because small files' download time is largely determined by latency. Path d offers the smallest latency (60ms) that helps reduce the file download time. To validate this, we reduce path c's RTT from 90ms to 10ms. Then Server 1 immediately outperforms Server 2 as indicated by the shaded blue bar.

Download large files. For large file download, usually both paths are saturated so the aggregated bandwidth is more important than the latency. In our case, although both servers have the same aggregated bandwidth ($a + c = b + d$), Server 1 still slightly outperforms Server 2, as illustrated on the right side of Figure 10. This is explained as follows. The aggregated bandwidth is dominated by LTE. The small LTE RTT offered by path a (compared to path b) shortens the TCP congestion control loop. This helps LTE recover from (real or more often, spurious [22])

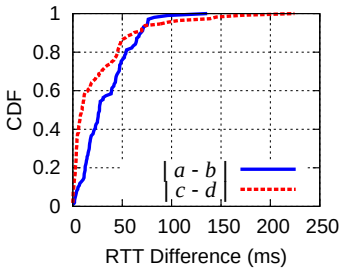


Figure 11: Distributions of path latency differences for 190 out of the Alexa top-500 websites. Refer to Figure 9 for the four paths.

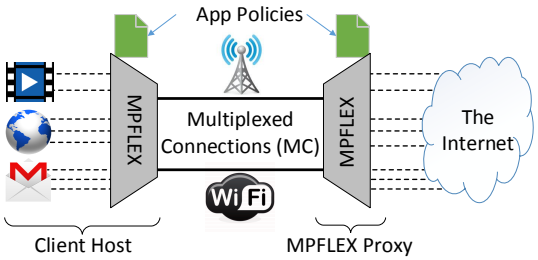


Figure 12: The MPFlex architecture.

losses more quickly, eventually leading to better LTE bandwidth utilization.

The right side of Figure 10 also illustrates an interesting phenomenon when we reduce path c's RTT from 90ms to 10ms for large file download. Doing so actually increases the overall download time non-trivially. In other words, improving a subflow in MPTCP may worsen the overall performance! We found this is attributed to the very design of MPTCP's RTT-aware scheduler. The very small RTT on a low-bandwidth path (in our case, WiFi) causes MPTCP to distribute more data onto that path. This can happen, for example, at the very beginning of a file transfer when both paths have empty spaces in their cwnd. As a result, less data is transferred over the high-bandwidth (LTE) path, leading to longer file transfer time.

The above emulation implies that choosing different CDN servers can lead to different network performance, depending on (1) MPTCP's scheduling algorithm, (2) traffic patterns, and most importantly, (3) characteristics of diverse paths. Within them, (1) and (2) are explained in the above emulation. We conduct another measurement to demonstrate the prevalence of (3), in order to show this is a real-world problem. For each of the Alexa top-500 U.S. websites, our phone sends DNS requests for its landing page over LTE and WiFi, and obtains the corresponding IP addresses of the web servers as IP_{LTE} and IP_{WiFi}, respectively. We found for 190 (38%) out of the 500 websites, their IP_{LTE} and IP_{WiFi} are different. They essentially correspond to CDN Server 1 and 2 in Figure 9, respectively. We then measure the RTT difference of path a and b, as well as the RTT difference of path c and d for each of these 190 sites.

The results are plotted in Figure 11. As can be seen, the RTT difference is more than 45ms for 20% of the websites, and the difference can be as high as 136ms and 224ms for cellular and WiFi, respectively. Recall that Figure 9 assumes the RTT difference is x=30ms. Note the RTT difference measured in Figure 11 is much larger than the RTT variation on the same path. Also note for some of the 190 websites, the servers associated with IP_{LTE} and IP_{WiFi} may be co-located for load-balancing purpose, and it is not easy to identify such websites. Therefore, Figure 11 is a conservative estimation of the path diversity.

Recommendations. We propose improvements to make CDN server selection multipath-aware. First, we can add *protocol support* allowing the CDN infrastructure to learn that a client has multiple network paths, by, for example, adding a backward-compatible extension to DNS protocol. The second and more important question is how to select the optimal server based on characteristics of all paths. Our previous discussion already sheds light on some high-level ideas. For small transfers, the goal is to minimize the delay. Therefore, among servers that are latency-wise closest to each of the client's IP addresses, the CDN can greedily pick the one with the minimum latency as long as that path has reasonable quality. For large data transfers, the goal is to maximize the overall bandwidth. If selecting different servers result in similar overall bandwidth, a possible strategy is to select server based on weighted sum for all paths' bandwidth (a path's weight is negatively correlated with its latency). Doing so shortens TCP's control loop and leads to higher throughput. We leave detailed design and implementation of a multipath-aware server selection scheme as our future work.

4. MPFLEX: A FLEXIBLE ARCHITECTURE FOR MOBILE MULTIPATH

Our measurements in §3 reveal several limitations of MPTCP: its interplay with short-lived flows can be further improved; it is important but currently difficult to incorporate application policies into MPTCP; the MPTCP extension is often blocked by middleboxes; MPTCP apparently does not work with other protocols such as UDP; MPTCP also incurs unexpected interaction with CDN.

We realize that many of these challenges stem from the origin of MPTCP, which was originally developed for improving the performance and robustness for datacenter networks [34]. Datacenters are "closed" ecosystems where latency is small, long-lived flows are common for intra-datacenter traffic, and administrators have full control over all applications and network elements. The mobile ecosystem, however, is drastically different, making naïvely porting MPTCP to mobile devices suboptimal.

4.1 The MPFLEX Architecture

Motivated by the above, we attempt to address an important research question: *what is a better system architecture for mobile multipath?* To this end, we designed, implemented, and evaluated a flexible software architecture of mobile multipath called MPFlex. As illustrated in Figure 12, MPFlex is a proxy-based solution. It is transparent to both client applications and servers, with several prominent features.

- MPFlex employs multiplexing to consolidate multiple (potentially short-lived) connections into two long-lived connections: one over WiFi and the other over cellular. The Multiplexed Connections (MC), shown in Figure 12, are persistent and are by default shared by all applications. They cover the "last-mile" links that are usually the bottleneck. This design overcomes MPTCP's key limitation when dealing with short-lived flows due to following reasons.

First, in MPFlex, since MCs are pre-established, an application connection (shown in dashed lines) only needs one handshake over usually the fastest path to inform the proxy to create the corresponding connection with the remote server, instead of

Table 3: Comparison of three multipath proxy solutions.

Multipath Solution	Multiplexing	Good Short Flow Performance	Protocol Applicability	User or Kernel	Transparent To SSL/TLS	Middlebox-friendly	Application Policies
MPTCP Proxy	No	No	TCP only	Mostly kernel	Yes	No	No
HTTP/2 Proxy + MPTCP	Yes	No	HTTP only	Kernel + user	No	No	No
MPFlex	Yes	Yes	Any protocol	Mostly user	Yes	Yes	Yes

handshakes for *every* subflow in MPTCP. This allows all subflows to be usable sooner, thus improving the bandwidth utilization. This handshake can be eliminated by applying the idea of TCP fast open [33], resulting in *zero-RTT handshake over multipath* between the device and the proxy.

The second benefit of multiplexing is, as an MC is persistent and long-lived, it can preserve the congestion window. This avoids the bandwidth probing (*e.g.,* TCP slow start). Note this advantage also exists in single-path, but it is more prominent in multipath by improving *all* subflows. When a long-lived MC has no data to transmit or receive, the radio interface switches to the IDLE state to save energy, while maintaining the TCP state.

Multiplexing has been employed by other application protocols such as SPDY [42], HTTP/2 [10], and QUIC [5]. MPFlex instead performs multiplexing at the transport layer while being transparent to upper-layer protocols. In particular, unlike a SPDY or HTTP/2 proxy that needs to be man-in-the-middle for SSL/TLS sessions (thus breaking the end-to-end security), MPFlex can transparently work with SSL/TLS.

- MPFlex *decouples* the high-level scheduling algorithm and the low-level OS protocol implementation. This is realized by implementing most of MPFlex's logic in the user space (unlike MPTCP's "all-in-kernel" approach), which obtains lower-layer information such as latency and congestion window size from kernel through a unified API. This dramatically simplifies the development and deployment of multipath features (§4.3).

- MPFlex is a flexible framework. Unlike MPTCP, MPFlex is provided as an OS service and can transparently provide multipath support for non-TCP protocols. An MC can be realized by a wide range of transport protocols such as TCP, reliable UDP, and SCTP [38], enabling continuous transport-layer innovation. Both features are realized through a *pair* of MPFlex modules deployed on both the client and the proxy. They not only perform (de)multiplexing, but also transparently intercept application packets (at the client), and serve as end points of MCs.

- MPFlex has visibility of all client traffic, making it an ideal vantage point for applying user-specified multipath policy based on application usage, performance, cellular data usage, and energy. Realizing such a policy framework by MPTCP itself is difficult unless a centralized traffic manager similar to MPFlex is introduced.

- MPFlex is middlebox-friendly. Since the multiplexing protocol runs *above* the transport layer, it does not require any network-layer or transport-layer extensions such as the MPTCP extension [18] that might not be recognized by today's firewalls and middleboxes.

Table 3 compares three multipath proxy solutions: MPTCP proxy, MPFlex, and HTTP/2 proxy with MPTCP.

4.2 MPFLEX Design and Implementation

We implement MPFlex as follows. Multiplexing is performed in a way similar to that in SPDY and HTTP/2, but across TCP connections instead of HTTP transactions (an approach similar to [31]). On the client side, uplink TCP data from applications is segmented and encapsulated into *messages*, which are then

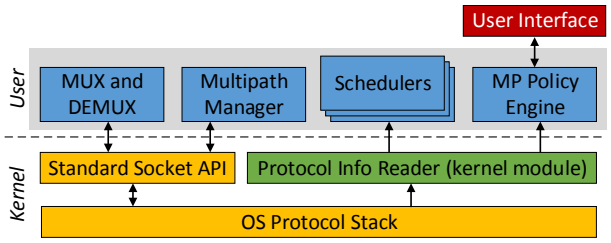


Figure 13: Components within an MPFlex endpoint.

distributed onto MCs. Each message has a small header containing its application connection ID, length, and message sequence number. Upon the reception of a message, the proxy performs demultiplexing by extracting the data and forwarding it to the remote server based on the connection ID. Downlink traffic is handled similarly but in the reverse direction. TCP SYN, FIN, and RST are also encapsulated into control messages to realize application connection management.

Based on the above basic multiplexing infrastructure, we developed MPFlex by adding three critical new components shown in Figure 13: multipath manager, schedulers, and policy engine. We currently implemented two types of MC: TCP and UDP (for multipath UDP). We only describe TCP here due to space constraints.

MPFlex is implemented in C++ with about 5K LoC on Nexus 5 with Android 4.4.4 as the client host and a commodity server as a MPFlex proxy. On the client side, we implemented a lightweight Linux kernel module using `netfilter` hooks to intercept uplink TCP packets and redirect them to the MPFlex userspace program, which manages MCs, application connections, and makes scheduling decision for uplink traffic. The MPFlex proxy performs similar operations for downlink traffic. Our implementation allows MPFlex to transparently provide multiplexing over multipath, so all applications can immediately benefit from MPFlex without modification.

Multipath Manager provides basic multipath support. At client host, MPFlex configures local routing tables, and sets up two regular TCP connections to the MPFlex proxy over the WiFi and cellular interface, respectively, as MCs. An MC is long-lived, unless its network interface is down. In that case multipath multiplexing falls back to single-path. The MC is reestablished when its interface becomes alive. Extending MPFlex for supporting more than two interfaces is also straightforward.

When an application connection issues a TCP SYN handshake, the client-side MPFlex module intercepts it, and sends a control message, containing the server IP and port, to the proxy-side MPFlex module over one MC selected by the scheduler (described below). The proxy then establishes the connection to the remote server. This differs from MPTCP's connection establishment where *every* path needs to perform its own handshake. A similar situation happens when closing the connection. The handshake message is treated as transport-layer payload, not bound to a particular

Table 4: Implementation overhead of different MPFlex plug-ins.

Plug-in for MPFlex	User-level LoC in C++
min-RTT Scheduler of MPTCP	70
Round-robin Scheduler of MPTCP	15
Buffer-aware Scheduler (§4.3, §5.4)	30
Smart reinjection (§4.3, §5.4)	60
Realization of a simple policy (§5.3)	20

path. Therefore, if WiFi is congested while LTE is active but less loaded, the handshake is performed over LTE. Also as mentioned before, MPFlex allows the handshake to be piggybacked with the uplink user data (of up to a threshold of n bytes) to achieve 0-RTT handshake over multipath.

Schedulers determine how to distribute data across multiple paths (MC in MPFlex). A key architectural design decision is to realize the scheduling logic at user level as much as possible. Toward this goal, we implement a small Linux kernel module that exposes a few in-kernel metrics such as RTT, TCP congestion window size, and TCP bytes-in-flight to the user space. The kernel module has very simple logic. It does not modify any state in the kernel, and remains unchanged once deployed in a specific kernel. We then build the actual schedulers in user space by utilizing the above kernel information API. Our design dramatically simplifies the scheduler implementation by *decoupling* the high-level scheduling algorithm and the low-level OS protocol implementation. In contrast, in MPTCP, they are tightly coupled, and upgrading MPTCP requires upgrading the entire kernel (tens of MB download). We have replicated two MPTCP schedulers: minimum RTT and round robin in MPFlex. Other schedulers (including those for non-TCP protocols) can be developed and plugged into MPFlex. To demonstrate the flexibility of MPFlex, we also designed two new schedulers to be described in details in §4.3.

Policy Engine provides a higher-level abstraction of determining when and how to use multipath according to user-defined policies. In our current implementation, a policy is an ordered list of rules specifying what kind of traffic should use which multipath scheme, such as "multipath with minRTT scheduler is only used by browsers and YouTube, and single-path is used in all other cases". User-defined policies are applied at a per-process basis. For a given traffic flow, MPFlex finds its corresponding process name using the methodology described in previous work [32]. For uplink traffic, the client-side MPFlex module can execute the policy by itself. For downlink traffic, instead of letting the proxy perform traffic classification, the client directly instructs the proxy on how to apply multipath by attaching a one-byte label to an uplink message. The proxy then applies the policy to the downlink traffic according to the label.

The policy engine can be extended to consider cellular billing (*e.g.,* disable multipath when the monthly data plan has less than 100MB left), energy (*e.g.,* disable multipath when the battery is low), and performance (*e.g.,* for YouTube, use cellular as the secondary path only when WiFi cannot provide 1Mbps throughput). We plan to realize such metrics in our future work.

4.3 MPFLEX Use Cases

Besides the performance benefit, a major advantage of MPFlex is flexibility. Developers can easily design multipath schedulers or realize custom policies at user-level by using a common API to get the kernel information. Such flexibility is demonstrated in Table 4, which summarizes the implementation efforts we made for different plug-and-play components to be discussed and evaluated in §5. Here we describe two examples in detail.

Buffer-aware scheduling. Recall in §3.1 that for small flows, MPTCP may perform worse than SPTCP in the following scenario. Suppose WiFi has a much smaller RTT than LTE. When WiFi's cwnd is fully utilized but LTE has available cwnd space, MPTCP's default scheduler *always* uses LTE regardless of its large latency. The optimal scheduling decision, however, is to buffer data at WiFi subflow's socket buffer unless it is full.

Inspired by this, we modify the min-RTT scheduler to let it consider both the network latency and the local buffering latency. Let $srtt$ be the (smoothed) RTT estimated by TCP. Recall the min-RTT scheduler picks a subflow that (1) has available cwnd space and (2) has the minimum $srtt$. Our modified scheduler, called TxDelay, instead picks a subflow that has the minimum $srtt + Q$ regardless of its cwnd status. Q quantifies how long it takes to drain the sender buffer. It can be estimated by $Q = \frac{B}{cwnd*mss/srtt}$ where B is the TCP sender buffer occupancy and mss is the TCP maximum segment size. It is worth highlighting that thanks to MPFlex's user-level realization, it takes only 30 lines of user level code to implement the TxDelay scheduler.

Smart Reinjection. Reinjection is a MPTCP feature allowing the same data to be sent over multiple subflows [21]. It helps reduce receiver side buffering due to out-of-order packets, leading to improved throughput when one or a subset of paths encounter performance degradation such as high loss rate or long latency caused by weak signal strength. MPTCP employs a static and fixed policy for reinjection: reinject packets when a subflow is terminated or its receiver buffer is full. We found for mobile multipath, MPTCP's default reinjection policy is often too conservative. For example, if a packet loss occurs on WiFi, the reinjection does not happen until the packet times out.

We propose to make the reinjection policy dynamic and configurable. For example, the sender can perform proactive reinjection based on different packet loss signals, such as duplicate ACKs and/or the receiver window occupancy (**`recvWin`**) embedded in the ACK packet. To realize this in MPFlex, the Protocol Info Reader (Figure 13) provides information such as **`recvWin`** and event callback such as TCP timeout and duplicate ACK, which are utilized by the user-level proxy to make smart reinjection decisions. A large **`recvWin`** indicates a large number of out-of-order packets are buffered at the receiver side, because packets with smaller sequence numbers (likely being transferred over the path with performance degradation) are not received. Therefore, reinjection of unacknowledged packets when **`recvWin`** is large helps eliminate the sequence number gap and thus improve the performance. We have implemented the following proof-of-concept reinjection policy. A message[3] transmitted on one path is reinjected to the other path in either of the two conditions: (1) its underlying packet experiences a TCP timeout, or (2) an ACK from the receiver indicates that the receiver buffer occupancy exceeds $\eta\%$ of the total buffer size. Note in case (2) each unique ACK triggers reinjection of at most one unacknowledged message, and η is a threshold determining the reinjection aggressiveness. We empirically set it to 75%.

5. EVALUATION OF MPFLEX

We conduct extensive evaluation of MPFlex. For performance, we compare MPFlex with MPTCP v0.89.5 (the latest version of MPTCP available for Android) with default settings and the same tunneling setup described in §2.1. All experiments were

[3]Recall in §4.2 that a message is the atomic transfer unit in MPFlex. We configure its maximum size to be the TCP MSS.

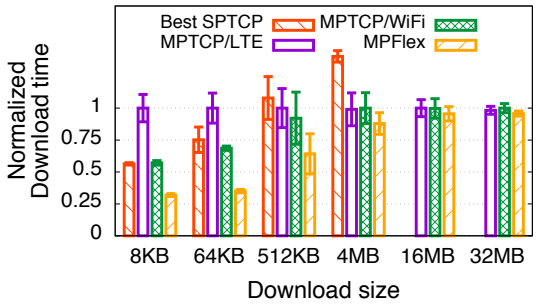


Figure 14: Single file download over MPTCP and MPFlex (best SPTCP results shown only for small downloads).

conducted using real WiFi and LTE networks on a Nexus 5 phone with Android 4.4.4. We use `tc` to apply bandwidth throttle and to add extra delay on both paths[4] based on recent large-scale measurements of metropolitan LTE [23] and WiFi [36] users. The same configurations were used by another recent MPTCP study [19]. For apple-to-apple comparison, MPFlex and MPTCP employ the same min-RTT scheduling algorithm, the same congestion control (decoupled Cubic *i.e.,* each path runs TCP Cubic independently), and the same proxy server unless otherwise noted. We next describe the evaluation results.

5.1 File Download

Download a single file. Figure 14 compares single file download time under three schemes: cellular-primary MPTCP, WiFi-primary MPTCP, and MPFlex, for different file sizes. Compared to the best MPTCP scheme, MPFlex reduces the download time by 11% to 49%, due to its simplified handshake procedure that makes better utilization of both paths.

Handling multiple short flows. We wrote a custom benchmark tool that generates small flows sequentially or concurrently. Figure 15 plots the overall download time for MPTCP and MPFlex under eight traffic patterns. Compared to downloading a single file, when handling *multiple* short flows, the advantage of MPFlex is more phenomenal with download time reduction ranging from 13% to 63%. As mentioned in §4.1, this is attributed to two features brought by MPFlex's strategic multiplexing: simplified handshake (same as the single file download case) and being capable of maintaining the congestion window[5] for multiple connections arriving in a bundle. We also note that the savings reduces a bit when the concurrency becomes higher due to improved bandwidth utilization of concurrent flows.

5.2 Web Browsing

How much performance gain can MPFlex offer under realistic applications and traffic patterns? To answer this question, we pick seven diverse websites and load their landing pages automatically with QoE Doctor [11] on Chrome browser on Nexus 5. To overcome frequent content change and server-side load fluctuation for some sites, we use Google Page Replay [1] to take a snapshot of each site, and host it on our replay server. To further make our setup realistic, we measure the RTT from proxy (MPTCP or MPFlex) to the real servers, and set the same RTT for the link between the proxy and our server when replaying each website.

The results are shown in Figure 16. Compared to Figure 15, the page load time (PLT) reduction is less, mostly due to the additional browser-side overhead (rendering page, parsing JavaScript *etc.*) and inter-object dependencies that often shift the bottleneck from network to local computation [41]. Nevertheless, the improvements are still impressive: compared to MPTCP proxy, MPFlex reduces the PLT by 7% to 20%. We also expect MPFlex will exhibit more advantages on newer mobile devices or tablets where computation is less likely to become the bottleneck.

5.3 Applying Multipath Policies

As described in §4.2, we have implemented a framework that allows applying different multipath policies at a per-process basis. We demonstrate its effectiveness of saving energy by conducting a case study as follows. We consider four apps: YouTube (playing a 150-second 1080p video), Skype (2-min VoIP call), Google Play (downloading a 50MB app), and Facebook Messenger (sending a message every 30 seconds). We enforce the following policy: use multipath for YouTube and Google Play, and single-path (WiFi) for Skype and Messenger. We compare the radio energy consumption of system-wide MPTCP (applied to all four apps) and MPFlex with the above application-aware multipath policy. As shown in Figure 18, MPFlex reduces the radio energy consumption for Skype and Facebook Messenger by 34% and 78%, respectively, while incurring no QoE degradation as explained in §3.3.

The policy framework can be extended to consider other factors such as billing and battery life. MPFlex can also enforce the policy at a finer granularity. Consider two use cases. (1) Let Chrome browser to use multipath only for `cnn.com`. (2) Disable multipath for all ad traffic. Both use cases can be realized by controlling multipath usage at a per-HTTP-session basis without modifying the apps. First, for an incoming HTTP session, the client-side MPFlex module obtains its associated domain name. This can be realized by examining directly the HTTP request or the TLS/SSL certificate for HTTPS. Second, it consults a local policy database to determine which multipath scheme to use. Third, the MPFlex client informs the proxy of the policy for this HTTP session using a small label attached to the multiplexed message (§4.2). We are currently implementing this feature.

5.4 Plugging-in Custom Schedulers

We evaluate the two MPFlex plugins described in §4.3.

Buffer-aware Scheduling. We evaluate the performance of our TxDelay scheduler by comparing it with the min-RTT scheduling algorithm. As shown in Figure 17, when WiFi RTT is much smaller than LTE RTT, TxDelay significantly outperforms min-RTT by reducing the file download time by 21% to 54%. TxDelay essentially makes multipath performs at least as well as single-path for small files (assuming RTT estimation is accurate). On the other hand, when RTT of both paths are similar, TxDelay exhibits similar performance as min-RTT (figure not shown).

Smart Reinjection. We compare our smart reinjection with the default MPTCP and MPFlex without reinjection. The workload is to download a 1MB file hosted at a server near the MPFlex proxy (4ms RTT between proxy and server, as to be justified in §5.5). We consider two network conditions: increased LTE latency (WiFi 10Mbps/50ms, LTE 9Mbps/150ms) and increased WiFi latency (WiFi 12Mbps/300ms, LTE 9Mbps/70ms). We use the public WiFi network in our office building. We made two observations. First, during the entire course of our experiment, the default min-RTT scheduler never triggers reinjection, which is done very conservatively as described in §4.3. As a result, MPFlex without

[4]WiFi: uplink 2020kbps, downlink 7040kbps, RTT 50ms;
LTE: uplink 2286kbps, downlink 9185kbps, RTT 70ms.

[5]Similar to a regular TCP connection, multiplexed connections in MPFlex still conservatively perform slow start after idle period.

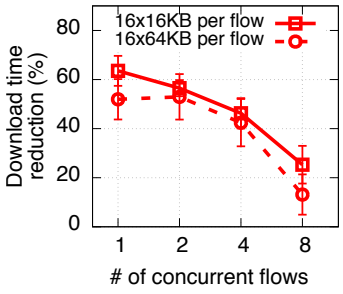


Figure 15: Transfer many short flows over MPTCP and MPFlex.

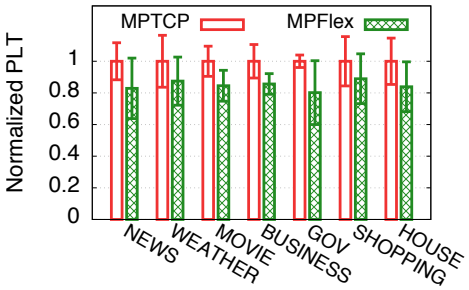


Figure 16: Fetch web pages over MPTCP and MPFlex.

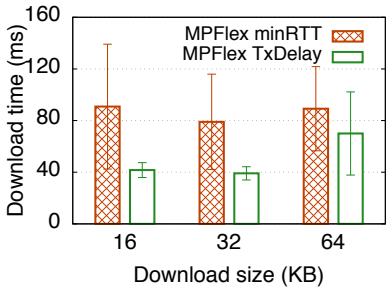


Figure 17: Performance of minRTT vs. TxDelay scheduler when the RTT difference between the two paths is large (20ms vs. 70ms).

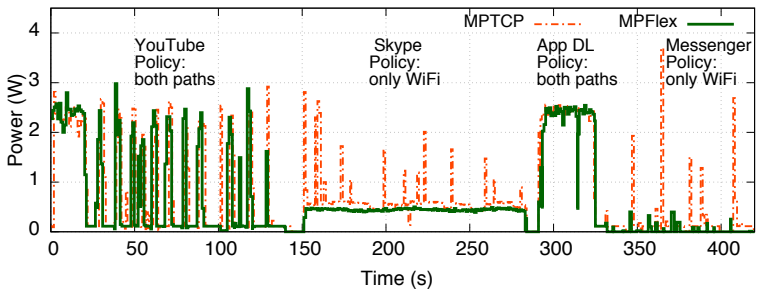


Figure 18: Case study: MPTCP applies multipath to all traffic, while MPFlex does that selectively based on user policy.

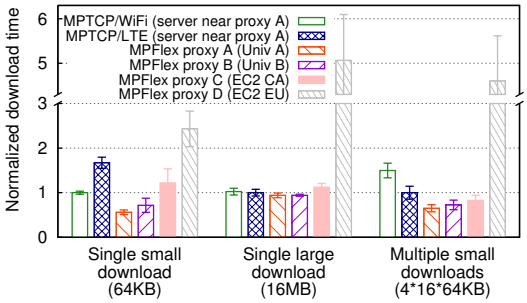


Figure 19: Performance impact of the MPFlex proxy location.

reinjection performs similarly compared to MPTCP. Second, smart reinjection reduces the overall download time by 10%±5% (over 12 runs) at the cost of reinjecting only about 1.5% of the total bytes, in both network conditions. The aggressiveness of reinjection can further be tuned by adjusting the buffer occupancy threshold.

5.5 Impact of Proxy Location

Our discussion in §3.4 indicates the location of a CDN server (in our case here, the MPFlex proxy) may affect the network performance in particular for multipath. To quantify this, we deployed the MPFlex proxy at 4 different locations referred to as A, B, C, and D. Their physical distances from our client smartphone are 3, 420, 3300, 6700 km, respectively. We also measured the minimum WiFi RTT between the mobile client and them to be 27, 44, 81, and 131ms, respectively, and the corresponding cellular RTTs are 49, 66, 100, and 148ms, respectively. We fix the RTT between the proxy and the server to be 4ms, by assuming the server is a nearby CDN node. A recently study measured the median RTT between a mobile carrier gateway and 30 popular content providers' servers to be ∼4ms [31]. We also evaluate a non-proxy configuration by letting the client directly connect to the server near Proxy A, which has the smallest latency from the client, using the default MPTCP.

We consider three workloads shown at the bottom of Figure 19. As shown, for small file download(s) whose performance is latency-sensitive, the proxy location matters. Nevertheless, despite being further away, Proxies B and C still achieve better or similar performance compared to the default MPTCP configuration, because the benefits of MPFlex outweigh the penalty of additional latency for B and C. Proxy D exhibits low performance because it is located at a different continent. The transoceanic link shifts the bottleneck from the last mile to the Internet. For a single large download, since bandwidth is more important than latency, Proxy A, B, and C exhibit very similar performance. Also, because the file size is large and no multiplexing is needed for a single application connection, MPFlex provides little benefit beyond what can be achieved by the vanilla MPTCP.

5.6 System Overhead

Despite being realized mostly at user level, MPFlex itself incurs negligible runtime overhead. We monitor CPU usage of a Nexus 5 phone with MPFlex enabled. Compared to the default MPTCP, no noticeable CPU usage increase was observed when downloading large files at high speed (∼30Mbps). The MPFlex protocol overhead, defined as the total message header size divided by the size of all transferred messages, is measured to be less than 1% across 20 Android apps we tested. Multiple instances of MPFlex proxy can be deployed in geographically distributed clouds to achieve scalability.

Since the MCs in MPFlex are long-lived, periodic keep-alive messages may need to be exchanged between a client and the proxy. Their periodicity should be no longer than the minimum NAT/firewall timeout of either path (usually the cellular path). To quantitatively measure the radio energy overhead incurred by MPFlex, we conduct an experiment on a Nexus 5 phone by sending keep-alive messages over the cellular pipes every 10 minutes for 24 hours. We use the hardware energy profiling interface (provided through the **`sysfs`** file system) of Nexus 5 to measure the energy consumption. We then compare the total energy with the scenario where MPFlex is not running for the same duration of 24 hours. The incurred radio energy is 0.18Wh, only about 2.5% of a typical smartphone's battery capacity. Since most cellular carriers' NAT/firewall timeout is longer than 10 minutes [43], the incurred energy overhead can be further reduced by increasing the keep-

alive periodicity.

6. RELATED WORK

Performance Characterization of Mobile Multipath. Although the potentials of mobile multipath have been known for a long time [9], it has recently become a hot research topic as fueled by smartphones and MPTCP. Chen *et al.* [13] studied performance of MPTCP over 3G/4G and WiFi. A similar studied was performed by Deng *et al.* [17] to compare the performance between single-path and multipath. Both studies focus on file download using controlled experiments. Han *et al.* [19] investigated how MPTCP helps improve web performance by in-lab experiments. They found SPDY [7] better interacts with multipath compared to HTTP/1.1, due to multiplexing. We make a further step by proposing a flexible transport-layer multiplexing infrastructure for multipath, which provides considerably more benefits than application-layer multiplexing does.

De Coninck *et al.* [15] conducted a measurement study of MPTCP involving 12 mobile users. They focus on transport layer characteristics of MPTCP such as RTT, retransmission, and reinjection. In contrast, our user study focuses not only on MPTCP itself, but also on the cross-layer interactions. We also combine passive and active measurements in our study to get a thorough understanding of applications' performance and energy utilization in the wild.

Multipath Energy. Some studies also examined the energy aspect of multipath. Nika *et al.* [26] characterized energy and performance of multipath in outdoor environments. Very recently, Lim *et al.* [39] improves MPTCP to make it energy-aware. Peng *et al.* [28] also proposes algorithms that tradeoff throughput performance and energy consumption for MPTCP. We leveraged the multipath power models derived by some of these work, and use them to characterize multipath energy consumption in real-world settings (§3.1).

Applications of Mobile Multipath. Besides improving existing applications' performance, MPTCP provides opportunities for enabling new use cases. Mobile Kibbutz [25] is a system allowing nearby users share their links with each other via multiple shorter range wireless links. MSPlayer [14] is a YouTube client that fetches multiple video sources over multipath to improve video experience. Croitoru *et al.* [16] leveraged MPTCP to achieve seamless mobility in WiFi by letting a client connect to multiple APs on the same channel. Our work complements these specific systems by characterizing and improving MPTCP itself. There are other systems dealing with multiple interfaces in general, such as energy-efficient interface selection [29] and utilization of concurrent WiFi APs [37].

7. CONCLUDING REMARKS

We make contributions to mobile multipath research in two aspects: crowd-sourced measurement and software architecture. The user study provides valuable insights of how well multipath works "in the wild". Based on the findings, we introduce new system concepts and research problems such as multiplexed multipath and multipath-aware server selection. We are working on a full-fledged multipath policy system by leveraging the infrastructural support of MPFlex, and plan to deploy it on the user trial to study how to derive good policies to improve application QoE over multipath while minimizing the energy overhead.

Acknowledgements

We thank the anonymous reviewers and our shepherd for their helpful feedback. This research was supported in part by NSF under CNS-1059372 and CNS-1345226, as well as by Indiana University Faculty Research Support Program (FRSP) – Seed Funding.

8. REFERENCES

[1] Google Web Page Replay Tool. https://github.com/chromium/web-page-replay.

[2] iOS: Multipath TCP Support in iOS 7. https://support.apple.com/en-us/HT201373.

[3] KT's GiGA LTE. https://www.ietf.org/proceedings/93/slides/slides-93-mptcp-3.pdf.

[4] OPTICOM, PESQ - perceptual evaluation of speech quality. http://www.opticom.de/technology/pesq.php.

[5] QUIC Protocol. https://www.chromium.org/quic.

[6] Shadowsocks Socks5 Proxy. https://shadowsocks.org.

[7] SPDY Protocol – Draft 3.1. http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3-1.

[8] S. Agarwal and J. R. Lorch. Matchmaking for Online Games and Other Latency-Sensitive P2P Systems. In *SIGCOMM*, 2009.

[9] P. Bahl, A. Adya, J. Padhye, and A. Walman. Reconsidering wireless systems with multiple radios. *ACM SIGCOMM Computer Communication Review*, 2004.

[10] M. Belshe, R. Peon, and M. Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, 2015.

[11] Q. A. Chen, H. Luo, S. Rosen, Z. M. Mao, K. Iyer, J. Hui, K. Sontineni, and K. Lau. Qoe doctor: Diagnosing mobile app qoe with automated ui control and cross-layer analysis. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 151–164. ACM, 2014.

[12] X. Chen, N. Ding, A. Jindal, Y. C. Hu, M. Gupta, and R. Vannithamby. Smartphone Energy Drain in the Wild: Analysis and Implications. In *SIGMETRICS*, 2015.

[13] Y.-C. Chen, Y.-S. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley. A Measurement-based Study of MultiPath TCP Performance over Wireless Networks. In *IMC*, 2013.

[14] Y.-C. Chen, D. Towsley, and R. Khalili. MSPlayer: Multi-Source and multi-Path LeverAged YoutubER. In *CoNEXT*, 2014.

[15] Q. D. Coninck, M. Baerts, B. Hesmans, and O. Bonaventure. A first analysis of multipath tcp on smartphones. In *17th International Passive and Active Measurements Conference*, volume 17. Springer, March-April 2016.

[16] A. Croitoru, D. Niculescu, and C. Raiciu. Towards WiFi Mobility without Fast Handover. In *NSDI*, 2015.

[17] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan. WiFi, LTE, or Both? Measuring Multi-homed Wireless Internet Performance. In *IMC*, 2014.

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

[19] B. Han, F. Qian, S. Hao, and L. Ji. An Anatomy of Mobile Web Performance over Multipath TCP. In *CoNEXT*, 2015.

[20] B. Hesmans, G. Detal, S. Barre, R. Bauduin, and O. Bonaventure. SMAPP: Towards Smart Multipath TCP-enabled APPlication. In *CoNEXT*, 2015.

[21] B. Hesmans, H. Tran-Viet, R. Sadre, and O. Bonaventure. A first look at real Multipath TCP traffic. In *International Workshop on Traffic Monitoring and Analysis*, 2015.

[22] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao,

S. Sen, and O. Spatscheck. An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance. In *SIGCOMM*, 2013.

[23] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *SIGCOMM*, 2014.

[24] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. SOCKS Protocol Version 5. RFC 1928, 1996.

[25] C. Nicutar, D. Niculescu, and C. Raiciu. Using Cooperation for Low Power Low Latency Cellular Connectivity. In *CoNEXT*, 2014.

[26] A. Nika, Y. Zhu, N. Ding, A. Jindal, Y. C. Hu, X. Zhou, B. Y. Zhao, and H. Zheng. Energy and Performance of Smartphone Radio Bundling in Outdoor Environments. In *WWW*, 2015.

[27] C. Paasch, S. Barré, et al. Multipath TCP in the Linux Kernel. http://www.multipath-tcp.org.

[28] Q. Peng, M. Chen, A. Walid, and S. Low. Energy Efficient Multipath TCP for Mobile Devices. In *MobiHoc*, 2014.

[29] T. Pering, Y. Agarwal, R. Gupta, and R. Want. CoolSpots: Reducing the Power Consumption of Wireless Mobile Devices Using Multiple Radio Interfaces. In *MobiSys*, 2006.

[30] F. Qian, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck, and W. Willinger. TCP Revisited: A Fresh Look at TCP in the Wild. In *IMC*, 2009.

[31] F. Qian, V. Gopalakrishnan, E. Halepovic, S. Sen, and O. Spatscheck. TM3: Flexible Transport-layer Multi-pipe Multiplexing Middlebox Without Head-of-line Blocking. In *CoNEXT*, 2015.

[32] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Profiling Resource Usage for Mobile Applications: a Cross-layer Approach. In *Mobisys*, 2011.

[33] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan. TCP Fast Open. In *CoNEXT*, 2011.

[34] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving Datacenter Performance and Robustness with Multipath TCP. In *SIGCOMM*, 2011.

[35] I. Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, (4):62–67, 2011.

[36] J. Sommers and P. Barford. Cell vs. WiFi: On the Performance of Metro Area Mobile Connections. In *IMC*, 2012.

[37] H. Soroush, P. Gilbert, N. Banerjee, B. N. Levine, M. Corner, and L. Cox. Concurrent Wi-Fi for Mobile Users: Analysis and Measurements. In *CoNEXT*, 2011.

[38] R. Stewart. Stream Control Transmission Protocol. RFC 4960, 2007.

[39] Y. sup Lim, Y.-C. Chen, E. M. Nahum, D. Towsley, R. J. Gibbens, and E. Cecchet. Design, Implementation and Evaluation of Energy-Aware Multi-Path TCP. In *CoNEXT*, 2015.

[40] P. Valerio. Using carrier wifi to offload iot networks. http://goo.gl/aQ15ii.

[41] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. Demystifying Page Load Performance with WProf. In *NSDI*, 2013.

[42] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. How Speedy is SPDY? In *NSDI*, 2014.

[43] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang. An untold story of middleboxes in cellular networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 374–385. ACM, 2011.

[44] X. Xu, Y. Jiang, T. Flach, E. Katz-Bassett, D. Choffnes, and R. Govindan. Investigating Transparent Web Proxies in Cellular Networks. In *PAM*, 2015.