MPBond: Efficient Network-level Collaboration Among Personal Mobile Devices

Xiao Zhu
University of Michigan
shawnzhu@umich.edu

Jiachen Sun
University of Michigan
jiachens@umich.edu

Xumiao Zhang
University of Michigan
xumiao@umich.edu

Y. Ethan Guo
Uber Technologies, Inc.
yhguo@umich.edu

Feng Qian
University of Minnesota
fengqian@umn.edu

Z. Morley Mao
University of Michigan
zmao@umich.edu

ABSTRACT

MPBond is an efficient system allowing multiple personal mobile devices to collaboratively fetch content from the Internet. For example, a smartwatch can assist its paired smartphone with downloading data. Inspired by the success of MPTCP, MPBond applies the concept of *distributed multipath transport* where multiple subflows can traverse different devices. We develop a cross-device connection management scheme, a buffering strategy, a packet scheduling algorithm, and a policy framework tailored to MPBond's architecture. We implement MPBond on commodity mobile devices such as Android smartphones and smartwatches. Our real-world evaluations using different workloads under various network conditions demonstrate the efficiency of MPBond. Compared to state-of-the-art collaboration frameworks, MPBond reduces file download time by 5% to 46%, and improves the video streaming bitrate by 2% to 118%. Meanwhile, it improves the energy efficiency by 10% to 57%.

CCS CONCEPTS

• **Networks** → **Network protocol design**; **Transport protocols**; • **Human-centered computing** → **Ubiquitous and mobile computing systems and tools**.

ACM Reference Format:
Xiao Zhu, Jiachen Sun, Xumiao Zhang, Y. Ethan Guo, Feng Qian, and Z. Morley Mao. 2020. MPBond: Efficient Network-level Collaboration Among Personal Mobile Devices. In *The 18th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '20), June 15–19, 2020, Toronto, ON, Canada.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3386901.3388943

1 INTRODUCTION

It is increasingly common that a user possesses multiple mobile devices. For example, despite being a full-fledged computer, a smartwatch naturally needs to pair with a smartphone; business people oftentimes carry two phones, one for work and the other for personal tasks [1, 5]; tablets bear large screens and reasonable portability, making them good companions of smartphones.

From the networking perspective, smart mobile devices are equipped with diverse network interfaces such as cellular, WiFi, and Bluetooth (BT), making them capable of communicating with remote Internet servers as well as other local devices. We make a key observation that despite such a mature wireless *hardware* support, the potential of the devices' network interfaces that can operate collaboratively is far from being fully exploited. In this paper, we bridge this critical gap by bringing networking *software* innovations to the smart mobile device ecosystem. Specifically, we develop MPBond, a holistic system allowing multiple personal mobile devices to collaboratively fetch content from the Internet. MPBond enables a wide range of use cases that today's mobile/wearable OSes do not support or provide optimal performance for:

• A smartwatch can assist its paired smartphone with downloading data over cellular (many COTS smartwatches today have direct cellular access). This leads to a much higher throughput compared to using a single device.
• WiFi networks offered by public places such as hotels often impose per-interface rate limit. Such a limit can be naturally overcome by multi-device collaboration since each participating device has its own WiFi interface.
• Two smartphones can share each other's LTE bandwidth. In other words, their cellular interfaces are "combined" by MPBond and can be used by apps as a single virtual interface.
• Wearables can be placed at a spot with good signal and act as WiFi/LTE "range extenders". When running low on battery, a smartphone can offload power-hungry LTE access to a smartwatch paired over an energy-efficient BT link.

By closely examining the above use cases, we notice that all of them can be realized under the *multipath transport* scheme, where user data can be distributed over multiple subflows (paths).

Unlike traditional multipath paradigms such as MPTCP [45], MPBond needs to support *distributed* multipath where subflows traverse different devices. Specifically, MPBond involves one *primary* device, where the client app runs, and multiple *helper* devices, which boost the primary's network performance. Without loss of generality, for the first use case above, the traffic from the primary is intercepted by the MPBond service, which distributes part of the traffic to the helpers over local wireless links (called *pipes*), and transmits the remaining over the primary's cellular interface. The helpers then forward the traffic to the remote server through their own cellular interfaces. The reverse (downlink) direction works in a similar way: the server or an MPBond-capable proxy distributes the content to the primary and helpers. The primary merges all the received parts and delivers the content to the client app.

The above scheme provided by MPBond appears to be intuitive. However, we face numerous challenges when designing and implementing the system. How to properly manage heterogeneous devices and local wireless links? How to strategically leverage the helper devices to improve the network performance? How to design a robust multipath scheduler that considers both remote paths and local pipes, with the latter being unique in MPBond? How to expose appropriate interfaces to users and applications? How to make the whole MPBond system transparent to client and server applications? We next highlight our key design aspects.

• As a distributed multipath transport framework, MPBond allows a subflow to traverse a helper, and enables helpers to exchange data with the primary device over pipes. We develop a scheme to flexibly manage the pipes using different wireless technologies such as WiFi and BT. To support distributed multipath and pipes, we extend MPTCP's control plane protocol to coordinate the primary and helpers (§4.1).

• MPBond splits any subflow into two TCP (sub)flows, one between the primary and the helper, and the other between the helper and the server. TCP splitting benefits end-to-end TCP sessions that span heterogeneous networks as the case of MPBond (the Internet and the pipes). More importantly, doing so allows buffers to be set up between the split flows, which effectively mitigate the negative performance impact incurred by the fluctuating network condition on either network. Although TCP splitting is not new [25, 51], we take this concept a step further by applying it to helper devices in the context of mobile multipath transport (§4.2).

• We develop a Pipe-Aware Multipath Scheduler (PAMS) that strategically distributes traffic onto multiple subflows. Tailored to MPBond, PAMS consists of three key components: (1) a subflow latency estimation module that considers pipes, helper-side buffering, and heterogeneous networks; (2) an algorithm that makes judicious scheduling decisions to ensure low delivery latency for each packet; and (3) a smart reinjection scheme that deals with fluctuating network conditions and possible failures over pipes (§4.3).

• MPBond allows users to flexibly specify various policies such as granting per-app usage permission, limiting per-device resource consumption, and prioritizing traffic (§4.4).

We implement MPBond on commodity mobile devices including Android smartphones and smartwatches. We showcase that most of MPBond logic can be implemented in the user space while maintaining full application transparency and good performance (§5). We then systematically evaluate MPBond over real mobile networks. Our key evaluation results consist of the following (§6).

• Compared to kibbutz [35] and COMBINE [13], two state-of-the-art systems, MPBond reduces the energy consumption by 10%-57% under a wide range of network conditions with various workloads (file download, video streaming).

• Under varying and in-the-wild network conditions, MPBond reduces the file download time by 13%-35% compared to kibbutz and COMBINE. The reduced download time also translates to lower energy consumption.

• We show the need of three collaborative mobile devices to deliver good QoE for bandwidth-hungry 360-degree video streaming. We also demonstrate the effectiveness of MPBond's dual mode.

Overall, MPBond is an efficient and practical system that innovates network-level collaboration among personal mobile devices through applying the concept of distributed multipath. Compared to other cross-device data sharing schemes [13, 35, 48], MPBond offers several advantages including better performance as boosted by the PAMS scheduler, application transparency, and more flexibility (§3.4). Also none of the above studies has considered or experimented using wearable devices. Our contributions made in this work consist of novel use cases, the MPBond design/implementation, and comprehensive evaluation in real-world settings. Note that MPBond is open-source on GitHub [11].

2 BACKGROUND AND RELATED WORK

Multipath Transport is a promising technique that simultaneously leverages multiple network paths to accelerate data transfers. MPTCP [46], the *de facto* multipath solution, brings a shim layer between the socket interface and multiple underlying TCP subflows. Operating at the transport layer, it requires no modifications to both applications and networks. MPTCP's value has been widely evidenced by the joint uses of different paths including WiFi and LTE [20], WiFi and WiGig [47], WiFi and Bluetooth [52], LTE and 5G [34], multiple WiFi links [17], multiple cellular carriers [30], and multiple datacenter network paths [45], as well as the QoE benefits to a variety of Internet applications such as video streaming [16, 24], web browsing [23], and interactive apps [19, 29].

As the core component of a multipath transport system, a packet scheduler distributes data onto different subflows established over potentially heterogeneous network paths. MinRTT [40] is the default scheduler of MPTCP, which selects the path with available space in congestion window and the minimum network RTT. There are also studies on innovating the scheduling algorithm design to improve MPTCP performance [21, 22, 29, 31, 47, 49]. Besides MPTCP, there exist other multipath transport protocols such as MPRTP [50], MPQUIC [18], MP-H2 [37], and MP-RDMA [33].

In above studies, all the subflows are established from a single host. MPBond instead distributes the subflows to traverse multiple mobile devices to facilitate the collaboration.

Multi-device Collaboration. Existing work have made various efforts to tackle the network-level collaboration among multiple devices to deliver the data. Mobile kibbutz [35] leverages tethering based MPTCP to enable a device to transmit network data through another tethered device's cellular network, while using its own cellular interface. Specifically, on Android, a device (denoted as P) can be tethered to another device (denoted as C) through either over USB cable or wireless network. When wireless tethering is used, Device P essentially acts as a WiFi AP (called SoftAP mode) that receives traffic from Device C (the WiFi client) over WiFi, and forwards it to Internet servers over P's own LTE network, for uplink traffic. The downlink direction operates in a similar manner. Device C now has two local interfaces: a WiFi interface tethered to P and its own LTE interface, upon which MPTCP can be applied.

PRISM [27] is a heavy-weight mechanism that strips a single TCP flow over multiple WWAN links by significantly modifying the TCP stack in the OS kernel and uses a custom proxy to utilize multiple WWAN connections of different devices.

Other network-level collaboration solutions focus on specific applications by functioning at the application layer. COMBINE [13] collaboratively download files from HTTP servers among wireless peers by using HTTP byte-range requests. MicroCast [26] targets at video streaming. Cool-Tether [48] proposes energy-efficient cellular tethering for web browsing traffic only.

MPBond instead is light-weight and offers several advantages including better performance as boosted by its judiciously designed scheduler, application transparency, and more flexibility. Also none of the above work has been applied to wearables. Besides network-level collaborations, there also exist systems [12, 38, 39] that share other I/O resources among multiple mobile devices.

3 MOTIVATION

3.1 Incentives to Carry Multiple Devices

It is increasingly prevalent that users possess more than one mobile device [7]. People oftentimes carry two smartphones due to various reasons. For example, one phone is used for work and the other is used for personal tasks – such a physical separation minimizes the likelihood of business data being leaked or compromised [1, 5]. Another important reason for carrying two phones with different carriers is that the carriers have complementary coverage [29] so that the user can switch between the devices to enjoy better performance. This is in particular popular in countries such as India where prepaid plans are prevalent [6]. People may also carry their old phones as portable WiFi hotspots [8], or have a second phone with a roaming-friendly sim card [4]. Other reasons for having two phones include mitigating battery anxiety, preventing theft, providing extra storage, and backing-up sensitive data locally [2]. Note that people do not explicitly buy two new phones; instead they typically use their old phones as a second device – around 46% of Americans upgrade their smartphone every two years or less [3].

Compared to carrying two phones, an even more popular trend is to wear a smartwatch while carrying a smartphone. In particular, many smartwatches such as Apple Watch Series 4 and Samsung Gear Frontier have built-in sim cards, allowing them to access cellular data networks as a typical smartphone does. In addition, there are many other common combinations of dual or triple mobile devices with Internet access capabilities, such as smartphone+tablet and smartphone+laptop+smartwatch. Despite their prevalence, the potential of the devices' network interfaces that are concurrently operational is far from being fully exploited.

3.2 Benefits of Multi-device Collaboration

A *network-level* collaboration among multiple devices can significantly improve the network performance. The basic idea is straightforward: when the last-mile wireless hop is the bottleneck (which is typically the case), having multiple devices download data simultaneously can effectively improve the overall WWAN-side (wireless wide-area network, *i.e.,* the Internet) throughput. Meanwhile, the content received by individual devices is merged over WLAN (wireless local-area network) and delivered to the application (§3.4).

We now experimentally demonstrate the benefit of WWAN-side throughput aggregation, by measuring the performance of concurrent multi-device data transfers, in particular when wearable devices are involved – few prior studies have investigated that to our knowledge. We consider three COTS mobile devices with each using a different cellular carrier, as detailed in Figure 1. We place them side-by-side (0.2 meters apart), and let them perform concurrent bulk download from a nearby server over their own LTE networks for 1 minute. On each device, we sample TCP throughput every 200ms. The experiment was conducted in three locations: a university office, a residential apartment, and a grocery store. Figure 1 plots the throughput distributions of different devices and their combinations. As we do not consider the WLAN-side merging step in this measurement, the overall throughput achieved by multiple devices is the sum of that for each individual device (*e.g.,* "AS" corresponds to the total throughput of A and S).

Figure 1 indicates that the three carriers exhibit different performance at the three locations, with median throughput ranging from 6.2Mbps to 61.8Mbps. Assuming WLAN-side merging is not the bottleneck, leveraging two interfaces improves the overall throughput by 15.8% to 474.2%, and simultaneously using three devices boosts the throughput by 63.1% to 695.4%, compared to using only a single device (interface). The aggregated throughput can effectively support bandwidth-hungry applications such as UHD video streaming, mobile VR [28], and mobile holographic communication [42]. We also notice that there is no device whose network performance constantly outperforms the other two in all three locations. Therefore, one can also dynamically choose the best device based on its live network condition in order to satisfy the app's minimum QoE requirement – this is supported by MPBond.

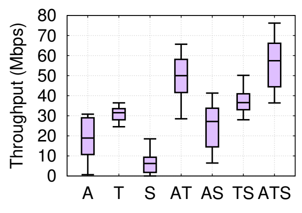
3.3 Networking Capability of Wearables

The experiment in Figure 1 involves an LTE-capable smartwatch. Despite its decent throughput, it may still raise concerns about its networking capability compared to full-fledged smartphones and tablets. We therefore experimentally compare the LTE throughput of two devices: an LG Urbane 2 smartwatch and a Nexus 6P smartphone, both released in the same year. To ensure fair comparisons, we repeatedly insert the same sim card (AT&T) into the two devices, and conduct 10 back-to-back TCP bulk download experiments on them over LTE at the same location, to understand the impact of the device capability on performance. The watch indeed achieves a statistically lower throughput compared to the phone: the smartwatch's median throughput (29.9Mbps) is only 53.3% of the phone's median throughput (56.1Mbps), likely due to the watch's small form factor that limits the antenna's size and tx/rx power. However, the absolute throughput values (median: 29.9Mbps, 90-th percentile: 31.4Mbps) indicate that commodity smartwatches' network interfaces can still contribute considerably to collaborative content delivery in particular when other devices' WWAN-side performance is poor. For example, Figure 1(a) and (c) show that the smartwatch (A) yields a higher throughput than the Sprint phone (S). Another potential concern regarding wearables is energy, which we will assess in §6.8.

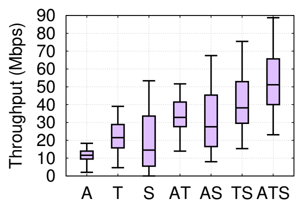
3.4 Do Existing Network-level Collaboration Schemes Suffice?

We summarize existing network-level collaboration schemes (§2) in Table 1. They suffer from several limitations as described below.

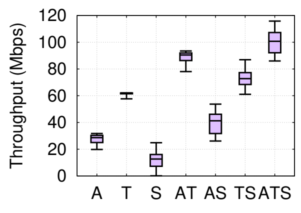
A Lack of Flexibility. A desired network-level collaboration must be flexible to support different types of applications and require minimal changes to the mobile and network infrastructure at the same time. From the application's perspective, a tethered device in kibbutz [35] performs as a simple layer-3 router, making



(a) Office



(b) Residence



(c) Grocery Store

Figure 1: Throughput distributions of different devices (carriers) and their combinations at 3 locations. (A: LG Urbane Watch 2 with AT&T; T: Pixel 2 smartphone with T-Mobile; S: Samsung Galaxy S9 phone with Sprint)

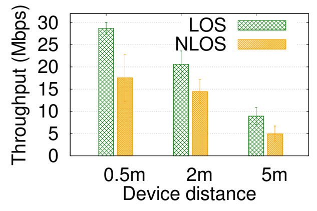


Figure 2: WLAN throughput from LG Urbane Watch 2 to Pixel 2 smartphone under different settings.

Table 1: Advantages of MPBond compared to existing systems designed for multi-device network-level collaboration.

Collaboration Scheme	System Layer	WLAN Layer	Application Transparency	Scheduling Consideration	Server-side Deployability	Mobile-side Deployability
MPBond	L4	**L4**	**Yes**	**WWAN + WLAN**	**standardized proxy**	**mostly userspace, > 2 mobile**
kibbutz [35]		L3	Yes	bottleneck	standardized proxy	kernel, up to two mobile
PRISM [27]		L3	Yes	bottleneck	server + new proxy	kernel, not available for mobile
COMBINE [13]	L5	L5	No	bottleneck	HTTP byte-range	userspace, > 2 mobile
MicroCast [26]		L5	No, video only	bottleneck	HTTP byte-range	userspace, > 2 mobile
Cool-Tether [48]		L3	No, web only	bottleneck	new proxy + byte-range	userspace, > 2 mobile

it difficult to flexibly support various enhancement and policies at layer-4/5 (§4.4). In addition, the tethering subsystem is usually tightly coupled with the OS/kernel, and is therefore difficult to be modified or extended. In Android, tethering has many practical limitations. For example, (1) tethering to more than one device is not supported, therefore, kibbutz supports at most two devices; (2) only one tethering connection (either WiFi or Bluetooth, but not both) can be established, hindering smooth handovers; (3) many carriers and devices lock the tethering feature or only provide limited data plan for tethering based hotspot. PRISM [27] relies on modified kernel TCP stack for both the sender and receiver and a custom PRISM proxy in the network, which incur significant deployment overhead. And its WLAN architecture relies on WiFi Ad-hoc mode, which is not available for Android and iOS smartphones and smartwatches. Other schemes including COMBINE [13], MicroCast [26], and Cool-Tether [48] require modification to the apps at layer-5 and the server must support HTTP byte-range requests for the network collaboration. Some of them are designed solely for a particular type of application traffic.

Suboptimal Performance due to Fluctuating Network Conditions. In kibbutz [35], the tethering approach, an end-to-end path consists of two segments: WLAN and WWAN. Figure 1 shows that the WWAN-side (LTE) throughput is indeed fluctuating. We next experimentally study the WLAN-side performance. Figure 2 shows the WLAN throughput when fetching data from an LG Urbane Watch 2 to a Pixel 2 phone with their physical distance varying under line-of-sight (LoS) and non-line-of-sight (NLoS) settings[1]. WLAN throughput varies significantly and is oftentimes *lower* than the WWAN throughput depicted in Figure 1. In other words, due to their heterogeneous link characteristics and complex interactions with the environment, WWAN and WLAN may exhibit highly different performance and *either* can become the bottleneck, in a very dynamic manner. The default tethering mechanism, however, often poorly deals with such heterogeneity and dynamics due to its simple layer-2/3 forwarding. For example, in tethering, the effective data rate is always the minimum of the WWAN and WLAN bandwidth; this can be improved by properly buffering data at the device that decouples the WWAN and WLAN. We will revisit this problem when describing MPBond's solution (§4.2). Besides, when making the scheduling decision, existing schemes only consider the performance of the bottleneck link between the WWAN and WLAN at a specific time, causing suboptimal workload distribution and hence the multipath performance.

Excessive Energy Consumption is one of the consequences of the suboptimal network performance. This is in particular an issue for wearable devices with small battery capacities. Even if the network condition is stable, in the tethering approach the congestion control is end-to-end, so the WWAN (WLAN) would be throttled to the WLAN (WWAN) bandwidth for data download (upload) when the WLAN (WWAN) is the bottleneck, causing prolonged WWAN (WLAN) radio-on time leading to increased energy consumption. Solutions that work at the application layer introduce idle network period between consecutive HTTP byte-range requests, lowering the energy efficiency.

4 MPBOND DESIGN

MPBond enables a user to jointly leverage her mobile devices to access the Internet in an application-transparent manner. As shown in Figure 3, MPBond involves two types of devices: one *primary* device and one or more *helper* devices (referred to as "primary"

[1]The distance between two devices can be large, *e.g.*,, when one device is charging or with another family member.

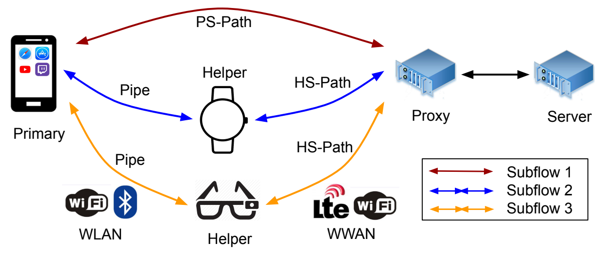


Figure 3: System Architecture of MPBond.

and "helper(s)" for brevity). We discuss the scenario of multiple primaries in §4.4.1[2]. The client application, such as a file downloader or a video player, only runs on the primary. TCP traffic from the app is transparently intercepted by the MPBond service and scheduled to transmit either over primary's own interface or through helpers with forwarding, i.e., different *subflows* shown in Figure 3. The reverse direction works in a similar way by distributing the traffic over multiple subflows from the MPBond-capable remote server and merging the content on the primary. To be fully transparent to Internet servers, the system can introduce an MPBond-capable proxy which hides MPBond from remote servers by establishing single-path connections with them. In the remainder of this paper, unless otherwise noted, the term "server" refers to either an MPBond-capable remote server or an MPBond proxy. Also, deployed as an OS service on the primary and helpers, MPBond is transparent to client-side apps as well.

We next describe how we address key design challenges of MPBond: How to properly manage subflows (§4.1)? How to overcome the limitations of the state of the art as described in §3.4 (§4.2)? How to intelligently distribute the traffic onto multiple paths while accounting for the heterogeneity between pipes and HS-Paths (§4.3)? How to properly interface MPBond with upper layers while considering various user-specified policies (§4.4)?

4.1 Subflow Management

The high-level concept of MPBond subflows is similar to that of MPTCP, except that (1) the subflows traverse different mobile devices, and (2) the primary and helpers need to perform local data exchanges to merge the received parts. We call the data channels between the primary and helpers *pipes*. We also denote the network paths between helpers and the server *HS-Paths* (Helper-server Paths), and the network path between the primary and the server (without a helper) the *PS-Path* (Primary-server Path). An end-to-end subflow therefore traverses through either a PS-Path, or an HS-Path and a pipe.

MPBond supports multiple concurrent pipes using different radio technologies such as WiFi and Bluetooth. The pipe is established by connecting the helper through WLAN to the primary which acts as a WiFi AP, or pairing the helper to the primary through Bluetooth. The scheduler dynamically selects a pipe by considering factors including performance, reliability, and energy efficiency, or simultaneously using multiple pipes to increase the data rate. We will revisit this feature in §4.4. Similar to MPTCP's subflows, the pipes can be flexibly torn down or established, and they are loosely coupled with a user TCP connection, allowing seamless migration among pipes without interrupting the data transfer.

[2]In this work, we assume the primary and all helpers are mutually trusted – the same assumption that other collaboration schemes make. Standard security primitives such as encryption and authentication can be applied to pipes to prevent attacks such as session hijacking and eavesdropping.

The overall handshake procedure in MPBond to establish a user TCP connection with subflows between the primary and the server leveraging helpers follows that in MPTCP, with additional control messages over pipes to coordinate with the helpers. Specifically, for the subflow involving an HS-Path and a pipe, the primary sends an INIT_MP_JOIN (INIT_MP) message with the necessary client and server information to the helper, allowing it to establish the second (first) subflow through an MP_JOIN (MP_CAPABLE) message. When the subflow is established, an MP_JOIN_OK or MP_OK message is returned to the primary as an acknowledgement.

Error Handling. MPBond should be robust to a wide range of errors. Compared to MPTCP, MPBond needs to further deal with pipes' failures. For example, on a subflow traversing through a helper, a failure of either its HS-path or its pipe will cause the subflow to be torn down and all its pending (unacknowledged) data to be reinjected to another subflow (§4.3.4). This ensures that no application data is lost due to either a WWAN or WLAN link failure.

4.2 Buffer Management and Helper-side Connection Split

MPBond maintains buffers at both end points (the primary and the server) to absorb network fluctuations and to accommodate the subflows' heterogeneous characteristics. Besides having these buffers, we make an important design decision of setting up buffers on helpers. Specifically, MPBond splits any subflow into two TCP (sub)flows, one between the primary and the helper, and the other between the helper and the server. The two flows thus cover the pipe and the HS-Path, respectively. Although TCP splitting is not a new idea [25, 51], we take this concept a step further by strategically applying it to helper devices, in particular wearable devices, in the context of mobile multipath transport.

Recall from §3.4 that when a helper is involved, the WWAN and WLAN exhibit vastly different link characteristics. TCP splitting can effectively improve the performance in such a scenario by shortening the TCP control loop [41]. More importantly, it allows buffers to be set up between the two flows. Such buffers effectively mitigate the negative performance impact caused by the bottleneck shift on a subflow. To illustrate this, consider a simple example where the pipe bandwidth increases due to the helper device being moved closer to the primary (Figure 2), causing the pipe's throughput (Th_{pipe}) becomes higher than that of the HS-Path (Th_{HS}). If there is a buffer at the helper, the buffered data can be transmitted at Th_{pipe} (instead of at Th_{HS} when there is no buffer), leading to a shorter data transfer time.

4.3 Pipe-aware Multipath Scheduler

As a critical component of a multipath transport system, a scheduler determines how to distribute the traffic onto multiple paths. There are several studies that improve the scheduler design in wireless settings [22, 29, 31, 36]. However, directly applying them to MPBond is difficult. First, most existing mobile multipath schedulers only deal with two paths (WiFi and cellular), and many solutions such

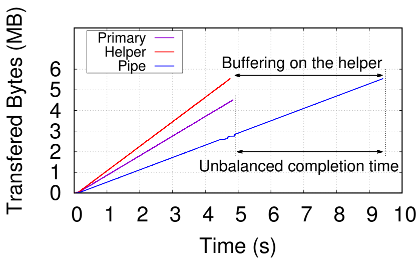


Figure 4: **Performance of MPBond configured with the minRTT scheduler.**

as [22] are inherently difficult to scale to more than two paths, which MPBond needs to handle. Second, none of prior studies considers the pipes, which are unique in MPBond.

We design a Pipe-aware Multipath Scheduler (PAMS) for MPBond. It differs from existing mobile multipath schedulers in two aspects: PAMS is capable of scheduling an arbitrary number of subflows, and it takes MPBond's TCP splitting and helper-side buffering (§4.2) into consideration when performing scheduling.

PAMS can be used by a wide range of applications such as file transfer, video-on-demand (VoD), web browsing, and cloud synchronization. All these applications involve transferring *data chunks* such as a file, a video chunk, an image, and a web page, which need to be delivered as fast as possible. Besides such chunked transfers, another type of traffic pattern is *real-time data streaming* such as live video streaming and low-latency gaming. Generally speaking, the benefits of multipath transport on these applications require switching the scheduling algorithm to a latency-favoring one such as [21] and [29], and blindly applying multipath to them may incur QoE penalty [36]. Designing a full-fledged scheduler tailored to the MPBond architecture for such latency-sensitive traffic is beyond the scope of this paper. Nevertheless, we provide easy-to-use interfaces (§4.4) for users to specify policies such as letting delay-sensitive traffic use single-path and giving it higher priority than other traffic so as to prevent potential latency inflation.

4.3.1 MinRTT Considered Harmful.

We first demonstrate the performance issue of directly applying the default minRTT scheduler. The primary establishes two subflows to a nearby server, one directly and the other through a helper. The downlink bandwidth of the PS-Path, the HS-Path, and the pipe are configured to be 8Mbps, 10Mbps, and 5Mbps, respectively. The primary downloads from the server a 10MB file using MPBond configured with minRTT. Figure 4 shows the download progress. Recall that MPBond maintains a buffer at the helper. As shown, on the positive side, due to TCP splitting and helper-side buffering, the bandwidth of all three paths is fully utilized. On the negative side, under the default minRTT scheduler, the two subflows cannot complete at the same time: the helper subflow finishes about 4.5 seconds later than the direct subflow. Note that in multipath transport, *simultaneous subflow completion* is a necessary condition for achieving the optimal download time [22]. This is because in the case where one subflow finishes earlier than the other subflow, the fast subflow can always "assist" the slow one, leading to an even reduced data transfer time.

The unbalanced subflow completion in Figure 4 is attributed to the fact that the scheduler, which runs at the server, only monitors the PS-Path and the HS-Path, and is unaware of TCP splitting mechanism and the downstream pipe. In other words, minRTT only tries to balance the completion time of the PS/HS-Path instead of the two end-to-end subflows. In this particular experiment, since the pipe bandwidth is lower than the HS-Path bandwidth, downlink data will be buffered at the helper and drained slowly over the pipe, leading to highly unbalanced subflow completion time.

A possible way of achieving simultaneous subflow completion is to modify the subflow availability condition: a helper subflow is considered to be available when the congestion window (CWND) of *both* the HS-Path and the pipe have available space (minRTT only considers the former). We implement this modification and find that it indeed almost achieves simultaneous subflow completion. However, by requiring an available CWND space for the pipe, this approach loses the capability of buffering at the helper, a key feature that MPBond should provide (§4.2). Therefore, the key challenge that PAMS should address is to *enable buffering at the helper while achieving simultaneous subflow completion.*

4.3.2 Deriving the Pipe-aware Delay (PAD).

We now describe the PAMS algorithm. We focus on the scheduler residing on the server for downlink traffic. We first derive the end-to-end (E2E) packet delay: at a given time T, if a packet is scheduled over a given subflow, how long does it take for the packet to arrive at the receiver (the primary)? Let B_s and B_p be the number of bytes buffered at the server and the helper, respectively, at T (they include both the TCP send buffer and the userspace buffer maintained by MPBond); let Th_{ps}, Th_{hs}, and Th_p be the measured downlink throughput of the PS-Path, the HS-Path, and the pipe, respectively; let OWD_{ps}, OWD_{hs}, and OWD_p be the one-way delay of the corresponding path. Given the above notions, the E2E delay for a direct subflow is $OWD_{ps} + \frac{B_s}{Th_{ps}}$, including both the propagation delay and the queuing/transmission delay. A subflow with a helper involves two buffers. It takes $T_1 = \frac{B_s}{Th_{hs}}$ to drain the server-side buffer. After T_1, the helper-side buffer level changes from B_p to $B'_p = \max\{0, B_p - Th_pT_1 + B_s\}$, which needs $T_2 = \frac{B'_p}{Th_p}$ to deplete. Therefore the overall E2E delay is $T_1 + OWD_{hs} + T_2 + OWD_p$. Plugging T_1 and T_2 into the above, we can derive the Pipe-Aware Delay (PAD) as:

$$\begin{cases} OWD_{ps} + \frac{B_s}{Th_{ps}}, & \text{if } i = 1 \\ OWD_{hs} + \frac{B_s+B_p}{Th_p} + OWD_p, & \text{if } i > 1, \frac{B_p}{B_s} + 1 > \frac{Th_p}{Th_{hs}} \\ OWD_{hs} + \frac{B_s}{Th_{hs}} + OWD_p, & \text{if } i > 1, \frac{B_p}{B_s} + 1 \le \frac{Th_p}{Th_{hs}} \end{cases}$$

where i is the index of the subflow (i=1 corresponds to the direct subflow). Th_{ps}, Th_{hs}, and Th_p can be estimated as an exponential weighted moving average of the ratio between CWND and RTT of the corresponding path. In practice, as directly measuring OWD is difficult, we approximate it using $\frac{RTT}{2}$. The second and third case in the above formula deals with $B'_p > 0$ and $B'_p = 0$, the two conditions considered by the max function when calculating B'_p.

4.3.3 The PAMS Algorithm.

PAD gives us an estimation of the E2E packet delay of a subflow. Now we consider how to use it to make scheduling decisions. A possible approach is to modify minRTT into "minPAD", *i.e.,* select the subflow with the minimum PAD as long as the subflow is idle (*i.e.,* the PS-Path or HS-Path has empty CWND space). Although this approach outperforms minRTT, it still tries to occupy all the HS-Path CWND space, thus may schedule more data than the subflow's actual capacity. We next show that it can

Algorithm 1: The Pipe-Aware Multipath Scheduler (PAMS).

```
Input: $S$ = A set of $N$ subflows. The algorithm executes when at least one
    subflow is idle, i.e., its PS-Path or HS-Path has empty space in CWND.
Output: Packet to transmit on a subflow $[packet, subflow]$.
1  $packet \leftarrow GetNextPacket()$;
2  $Th[1..N] \leftarrow GetSubflowThroughput()$;
3  $PAD[1..N] \leftarrow GetPipeAwareDelay()$;
4  $Idle \leftarrow GetIdleSubflows()$;
5  $Busy \leftarrow GetNonIdleSubflows()$;
6  $target \leftarrow GetIdleSubflowWithMinPAD()$;
7  $Diff \leftarrow 0$;
8  for each subflow i in Busy do
9      if PAD[i] < PAD[target] then
10         $Diff += (PAD[target] - PAD[i]) \times Th[i]$
11 if Diff > GetUntransmittedSize() then
12     return NULL;
13 else
14     return [packet, target];
```

be further improved through strategically *deferring* the scheduling to make more judicious scheduling decisions.

Let us consider two cases that require different scheduling strategies. First, when the server has a *large* amount of remaining data in the meta buffer[3] to send, it is important to improve the overall bandwidth utilization (*i.e.,* throughput) by keeping all the subflows busy. In this case, PAMS applies minPAD: as long as there are any idle subflows, the one with the minimum PAD will be immediately selected and made busy. The second case is that when there is only a *small* amount of remaining data, ensuring low-latency delivery and simultaneous subflow completion time is more important than maximizing the throughput. In this case, even when there is an idle subflow, PAMS may skip it (*i.e.,* deferring the scheduling) when there are non-idle subflows that can shorten the delivery latency.

Following the above idea, we develop the PAMS algorithm listed in Algorithm 1. As shown, *Idle* and *Busy* are the set of idle and non-idle subflows, respectively, and *target* is the idle subflow with the minimum PAD. Line 8 to 14 is the core part of the algorithm. It determines if it is possible to deliver all to-be-scheduled bytes in the meta buffer (or an application-defined data chunk, see §4.4) over currently busy subflows before the *target* subflow completes. For a given busy subflow i, its current buffered data will be drained in $PAD[i]$ time units, so the time budget allowing it to deliver additional unscheduled data before the *target* subflow completes is $PAD[target] - PAD[i]$. The throughput of the subflow i, $Th[i]$, is the minimum of the HS-Path throughput and pipe throughput when $i > 1$. The total number of unscheduled bytes that can be delivered by all busy subflows before the completion of the *target* subflow is therefore calculated as $Diff$ (Line 10). If such bytes are more than the total number of unscheduled bytes, we defer scheduling the current packet, allowing it to be later scheduled over a currently busy subflow (Line 12). Doing so will shorten its delivery time and facilitate simultaneous subflow completion. Otherwise, we immediately schedule the packet over the *target* subflow to ensure high bandwidth utilization (Line 14). Note that Algorithm 1 is for the downlink traffic, and the scheduler for uplink traffic is developed in a conceptually similar manner.

[3]In multipath transport, the (sender-side) meta buffer stores data passed from the application but is not yet scheduled. The meta buffer is different from the per-subflow buffer (B_s and B_p), which contains data that has already been scheduled to a subflow.

4.3.4 Data Reinjection.

In multipath transport, reinjection is a mechanism where data that has already been scheduled over one subflow (A) is "reinjected" into another subflow B. This may occur when, for example, A experiences unexpected performance drop or failure, or B's capacity suddenly increases. MPTCP employs a conservative and fixed reinjection policy where packets are reinjected only when their associated subflow is terminated or the receiver buffer is full. In MPBond, the involvement of multiple devices, heterogeneous networks, and helper-side buffering makes the network performance potentially more dynamic and fluctuating, thus necessitating more judicious reinjection decisions.

We next describe MPBond's reinjection scheme by detailing when, who, and how to perform reinjection. Specifically, a reinjection is triggered when there are no unscheduled bytes and $\frac{\text{maxPAD}-\text{minPAD}}{\text{minPAD}} > \eta$, where minPAD and maxPAD are the minimum and maximum PAD values across all subflows, respectively, and η is a parameter. The rationale is as follows. Ideally, all subflows' PAD should be similar, as PAMS implicitly controls the buffer levels (B_p and B_s) of the subflows to facilitate simultaneous subflow completion (§4.3). When some subflows' PAD becomes too large or too small, it implies severe fluctuations of their network performance. It is therefore the right time to launch a reinjection for promptly rebalancing the subflows. Regarding η, it determines the aggressiveness of the reinjection: reducing η incurs more frequent reinjections at the cost of increased bandwidth utilization. We empirically set η to 20%.

When a reinjection is triggered, MPBond moves up to r unacknowledged bytes with the highest sequence numbers from the subflow with maxPAD back to the meta buffer[4]. We calculate r as $(\text{maxPAD} - \text{minPAD}) \times B$ where B is the effective throughput of the subflow with maxPAD. Intuitively, r is determined in such a way that a slow subflow can catch up with the fastest subflow in terms of PAD. These r "recalled" bytes are scheduled again by PAMS.

4.4 User/App Interfaces and Policy Engine

MPBond provides 2 types of interfaces to users and app developers, respectively. First, it has a built-in console on the primary. This allows users to pair/unpair with helpers, manage the pipes, grant apps permission to use MPBond, monitor the devices' runtime status, and configure various policies (see below). In addition, MPBond exposes APIs allowing 3rd-party apps to programmatically use its service. The APIs include device/pipe management, status query of devices/pipes, callback functions of important events such as a change of the pipe configuration, and marking the boundaries of application data chunks[5]. Note that using such APIs is optional: MPBond is fully transparent to apps; the APIs just provide more fine-grained manipulation and detailed monitoring of MPBond. For example, based on its data rate, an app can dynamically switch between pipes with different bandwidth (*e.g.,* WiFi vs. Bluetooth), to reduce the energy footprint while meeting the QoE requirement.

[4]Data residing in the user-space buffer can be directly moved; data that stays in the kernel-level buffers will become redundant.

[5]By default, the *GetUntransmittedSize()* function in Algorithm 1 returns the total size of the to-be-sent data in the meta buffer. Developers can optionally define application-layer data chunks to make *GetUntransmittedSize()* return the remaining bytes of the current data chunk. This will expedite the delivery of each data chunk as opposed to all data in the meta buffer as a whole.

MPBond allows users to flexibly specify various policies. Our current prototype supports the following policies. (1) *per-app whitelist.* MPBond takes a "whitelist" approach: users need to explicitly grant permissions to apps and specify a (super)set of devices/pipes that the app can access through MPBond. Typically this is a one-time effort, provides good flexibility, and boosts security. (2) *Resource Usage.* MPBond allows disabling a device (either helper or primary) when its battery level drops below a threshold or its monthly cellular data usage reaches a pre-defined cap. (3) *Prioritization.* Users can configure rules that prioritize certain applications.

4.4.1 Dual Mode in MPBond. MPBond allows a device to have dual roles of both a primary and helper, and multiple primary devices may co-exist. We call this the *dual mode*, which enables the collaborating devices to better utilize their collective bandwidth in particular when the primary devices generate traffic at different time. Consider the following use case. Two close friends are watching different DASH videos, but each one's individual device may not provide sufficient bandwidth for its own video. To overcome this limitation, the two devices can be paired up using the dual mode: each device acts as the primary by fetching its own video, and meanwhile also as the helper by delivering the content for the other device. Since the two devices usually do not fetch video chunks simultaneously, the video chunks requested by each device can be downloaded over the two subflows with small probabilities of competing for the bandwidth with video chunks requested by the other device. This leads to improved QoE for both users.

In our current prototype, we realize the dual mode by running multiple independent instances of MPBond, as either a primary or a helper, on the same device. A limitation of this approach is that each MPBond instance independently makes scheduling decisions, which may be suboptimal due to a lack of global view of the network condition and traffic patterns. This issue can be addressed by introducing a lightweight "global manager" that coordinates all MPBond instances [14, 41]. We leave this as future work.

5 IMPLEMENTATION

We implement MPBond on commodity Android smartphones and Wear OS smartwatches. To support real-world evaluations with commercial Internet servers that may not support MPTCP and middleboxes that may block it, we implement a multipath TCP proxy in C/C++, following the methodology in [22]. Our implementation of MPBond consists of 8K lines of C/C++ and Java code excluding the base proxy system and is accessible on GitHub [11].

On the primary, most of the logic lies in a userspace MPBond service. It establishes the PS-Path (pipe) connections with the MPBond proxy (helpers). To support unmodified applications, we built a lightweight kernel module using *netfilter* hooks that intercept and redirect client application traffic to the MPBond service. WiFi pipes are implemented as long-lived TCP connections between the primary and helpers. We also implement Bluetooth pipes by leveraging the Android BluetoothSocket APIs to establish RFCOMM connections. The MPBond helper module is implemented in the userspace for the ease of deployment. It establishes HS-Path (pipe) connections with the proxy (primary). For each subflow, we use a circular queue to buffer packets in the userspace. These buffers work with the in-kernel send/recv buffers of the HS-Path and pipes together to achieve the performance and energy benefits.

A pipe is a long-lived data channel over which multiple user TCP connections are multiplexed. To do this, we add a tiny header before the application payload containing the TCP connection ID, message length, and sequence number to identify individual TCP connections. PAMS is implemented as a userspace scheduling module plugged into the MPBond proxy. The PS-Path and HS-Path information is obtained at the server by leveraging Linux *getsockopt* API. Pipe throughput is measured on the primary and sent to the helper through an encapsulated control message. We implement a flexible interface for a helper or the primary to determine when and which pipe's information to send to the proxy. In §6.2 we demonstrate how this flexibility can be helpful instead of fixing the feedback mechanism. Currently we use an out-of-band UDP channel to carry pipe-specific information over the return path of HS-Path/PS-Path for the sake of prototyping. In the future we will replace it with TCP options that can be integrated to the HS-Path/PS-Path ACKs. User-defined policies are enforced at a per-process basis. The MPBond services on the primary looks up the process name of a given flow by following the methodology in [43].

6 EVALUATION

We extensively evaluate MPBond under various network and device settings using synthetic and real apps to show the benefit of network-level collaboration. We examine the effectiveness of key design choices of MPBond through micro-benchmarks. We quantitatively compare MPBond with kibbutz [35] and COMBINE [13], the two major state-of-the-art solutions in Table 1, on network performance, energy consumption and app QoE using commodity smartphones and smartwatches over real LTE and WiFi networks.

6.1 Experimental Setup and Methodology

Our proxy supporting both MPBond and our implementation of kibbutz [35], which employs tethering-based MPTCP, runs on a commodity Ubuntu 16.04 server with 4-core 3.6GHz CPU and 16GB memory. The proxy uses the decoupled CUBIC as the congestion control algorithm (*i.e.,* each path runs TCP CUBIC independently). The server hosting files and video contents is in close proximity to the proxy, and the path between them has very high network bandwidth, not being the bottleneck of the end-to-end paths. For COMBINE [13], as no proxy is required, multiple mobile devices send HTTP byte-range requests directly to the server to fetch the chunks of different ranges in the same object. The requests are scheduled by a work-queue algorithm that sequentially downloads chunks on each path and returns them to the primary device. For COMBINE, we use a default chunk size of 256KB. For small file download (*e.g.,* 512KB) we also try two smaller chunk sizes (128KB and 64KB) and report the best performance. By default, MPBond uses PAMS as the multipath scheduler.

Our mobile devices include a Pixel 2 phone, a Nexus 6P phone, and an LG Urbane 2 smartwatch. We perform evaluation of MPBond using both emulated and real network conditions. To emulate certain network conditions, we use Linux `tc` to throttle the bandwidth on real WWAN and WLAN, while capturing the latency dynamics from commercial wireless networks. We also conduct experiments using real LTE networks at different places. To understand the impact on battery, we use full-fledged energy models [15, 32] to estimate the energy consumption incurred by network transfers.

6.2 Microbenchmarks

We start with examining the key design choices of MPBond. We focus on a two-device setting where a Pixel 2 with T-Mobile acts as the primary and a Nexus 6P with AT&T acts as the helper.

Benefit of Helper-side Connection Split. One key design aspect of MPBond is to decouple the HS-Path and the pipe with a buffer on the helper (§4.2), to absorb network fluctuations and accommodate heterogeneous subflow characteristics. To understand its impact on energy and performance, we compare MPBond with kibbutz, which does not incorporate this design, using various fixed scheduling ratio on subflows – transmitting $p\%$ of 4MB file over the PS-Path and $1 - p\%$ over the HS-Path and the pipe. We also derive the optimal ratio offline from an exhaustive searching of p.

We start with stable network condition where the bandwidth of both PS-Path and pipe are 5Mbps, and the bandwidth of HS-Path is 10Mbps. Figure 5 shows that MPBond reduces energy consumption by 10%-22% compared to kibbutz using fixed scheduling ratio, while achieving almost the same download time. This is because helper-side connection split allows the transmission on HS-Path to finish much earlier than that on the pipe, reducing the LTE radio-on time.

We then study the performance and energy impact under a changing network condition. We start from the stable profile described above, and after 2s, drop the bandwidth of HS-Path to 1Mbps. Figure 6 shows the download time and energy consumption of downloading a 4MB file. Both the download time and energy consumption are reduced when helper-side connection split is in effect. The improvement is much higher when more data is scheduled to the HS-Path and the pipe at the time of sudden bandwidth drop. This indeed confirms that the buffer between the split flows absorbs the fluctuation of network condition.

Benefit of Flexible Feedback. MPBond allows pipe information to be shared over the multiple HS-Paths and/or the PS-Path (§5). A simple yet effective policy of sharing such information for the 2-device case is to send the pipe's information over both the HS-Path and the PS-Path, when there's data transfer on the corresponding path. We call this policy "flexible" and compare it with sending the pipe feedback over the HS-Path only, with different timings: (1) when there's data transfer on either HS-Path or pipe ("fixed-always"), and (2) when there is data transfer on HS-Path only ("fixed-on-demand"). We run an experiment of downloading a 4MB file. Initially, PS-Path=5Mbps, pipe=5Mbps, and HS-Path=10Mbps. After 2s, pipe bandwidth increases to 10Mbps. As Figure 7 shows, "fixed-always" inflates the energy consumption since it keeps the helper's radio active by sending information feedback even if there is no data transfer on the HS-Path. "Fixed-on-demand" mitigates the issue by sending the feedback only when there is data transfer on the HS-Path. However, it still incurs performance degradation as the pipe information is not up to date. Instead, the "flexible" policy keeps sending feedback over the *PS*-Path when there is no transfer on the *HS*-Path, keeping the pipe information updated without waking up the helper's radio, thus improving both performance and energy consumption.

Estimating Pipe Buffering. PAMS estimates the pipe buffering time of a packet based on the buffered data on the helper and the pipe throughput ($\frac{B_p}{Th_p}$) (§4.3.2), instead of directly measuring the packet buffering delay incurred by the helper, *i.e.,* the time between when a packet arrives at the helper and when it comes out, which may not be up-to-date. To demonstrate the advantage of such approach, we conduct file downloads of 4MB under a stable network condition: the PS-Path and pipe bandwidth are 5Mbps, and the HS-Path bandwidth is 10Mbps. With the estimation based on the buffered data on the helper and the pipe throughput, the file downloads take 3.6s on average, compared to 4.4s on average using direct buffering delay measurements. The suboptimal performance of the latter approach is due to the fact that the scheduler always receives the stale buffering delay measurements which are inaccurate, thus making the scheduling decisions suboptimal.

Reinjection Under Changing Network Condition. Reinjection in MPBond helps to reduce the download time under changing network conditions (§4.3.4). To examine the effectiveness, we download a 4MB file under a changing network condition. At the beginning of transfer, the PS-Path and pipe bandwidth are 5Mbps, the HS-Path bandwidth is 10Mbps. After 2s, pipe bandwidth drops to 1Mbps. When reinjection is enabled, the download time is 4.8s, 49% of the download time without reinjection (9.7s). The improvement is attributed to the data on the slower pipe being reinjected to the PS-Path so that the pipe transmission can catch up.

6.3 Stable Network Conditions

In this section, we evaluate the performance and energy efficiency of MPBond under stable network conditions. The workload is downloading files with sizes ranging from 512KB to 2MB. We vary the number of mobile devices from 1 to 3 and measure the download time and energy consumption. We also study MPBond-Naive, another variant of MPBond where the default minRTT scheduler instead of PAMS is used. For each test, we repeat the download 20 times and report the mean value and the standard deviation.

Figure 9 shows the results under a common network bandwidth setting. Each plot in Figure 9 has 7 clusters corresponding to different schemes with different number of devices. A cluster that is closer to the bottom left has a lower energy consumption and a shorter download time. Note that when calculating the energy consumption, we consider all the mobile devices involved. Compared to kibbutz, MPBond reduces the download time (energy consumption) by 5%-11% (10%-14%), when there are 2 devices. When the number of devices becomes 3, compared to kibbutz which cannot utilize the extra device due to its architectural limitation, MPBond improves the download time by 25%-30% while maintaining a similar total energy consumption. Compared to COMBINE, MPBond brings even higher improvements in terms of both download time and energy consumption. With two (three) devices, MPBond improves the download time by 15%-21% (12%-26%), and reduces the energy consumption by 28%-38% (22%-25%). While MPBond-Naive has a similar energy consumption compared to MPBond, it sacrifices the performance due to suboptimal scheduling decisions that lead to imbalanced subflow completion (§4.3). These improvements are attributed to multiple design choices of MPBond including the system and pipe realization at Layer 4, the helper-side connection split and buffer, as well as the carefully designed multipath scheduler.

To better understand the impact of using more devices, we further break down the total energy consumption for different schemes in Figure 10. MPBond-Naive is omitted here for brevity. As shown, using more devices does increase the total energy consumption,

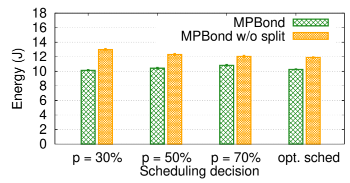


Figure 5: Energy benefits of split under stable network conditions.

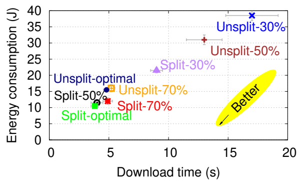


Figure 6: Performance and energy benefits of split under changing network conditions.

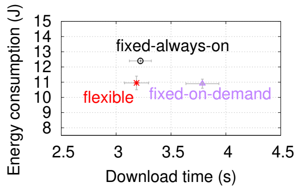


Figure 7: Performance and energy consumption for different feedback mechanisms.

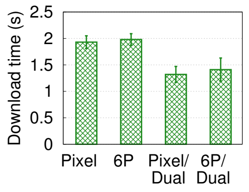


Figure 8: Dual Mode reduces download time.

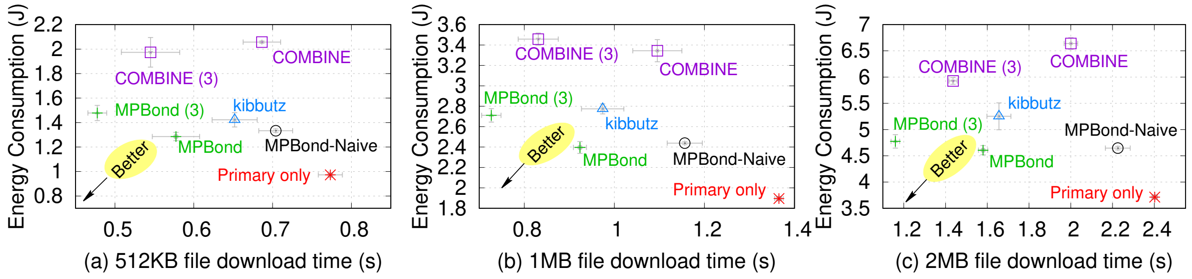


Figure 9: Bulk download performance under stable network condition (PS-Path: 8Mbps, HS-Path: 10Mbps, pipe: 5Mbps): Single device (Pixel2), MPBond/COMBINE w/ 2 devices (Pixel2+Nexus6P), MPBond/COMBINE w/ 3 devices (Pixel2+Nexus6P+LG2), and kibbutz (Pixel2+Nexus6P).

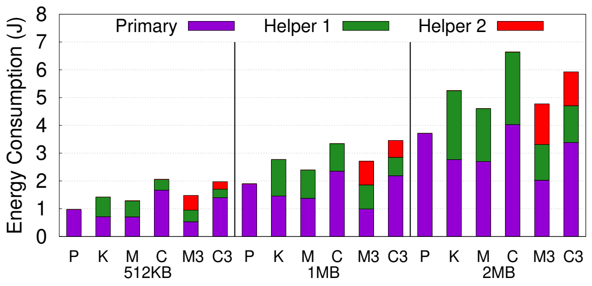


Figure 10: Energy breakdown of different schemes under stable network condition (PS-Path: 8Mbps, HS-Path: 10Mbps, pipe: 5Mbps): Primary only (P), kibbutz (K), MPBond (M), COMBINE (C), MPBond w/ 3 devices (M3), COMBONE w/ 3 devices (C3).

and COMBINE even increases the energy of the primary due to its poor scheduler design that does not distribute the workload in an efficient manner. MPBond instead reduces the energy on the primary when more devices are used, while keeping a reasonably higher total energy consumption. Compared to kibbutz, the energy improvement of MPBond goes mostly to the helper device thanks to its buffering strategy that reduces radio-on times.

To more systematically understand the benefits of MPBond against kibbutz and COMBINE, we further carry out experiments under more bandwidth combinations. We focus on the 2-device case where we use Pixel 2 as the primary and Nexus 6P as the helper, with the pipe bandwidth limited at 5Mbps. We first examine the energy improvement of MPBond over kibbutz, with different PS-Path and HS-Path bandwidths. Figure 11 plots the energy saving results. With higher HS-Path bandwidth and lower PS-Path bandwidth, MPBond's energy benefit is maximized: for 1Mbps PS-Path and 18Mbps HS-Path, energy consumption is reduced by 31%, 37% and 47% for 512KB, 1MB, and 2MB download, respectively, while the download time of MPBond is slightly better than kibbutz. The energy savings mainly come from the effectiveness of helper-side buffering that reduces the radio-on time of the faster link under heterogeneous WWAN and WLAN links. We then compare MPBond with COMBINE by changing the PS-Path bandwidth. Figure 12 plots the download time for both schemes. As shown, MPBond reduces the file download time by 14%-46%, leading to energy savings of 24%-57%. Overall, as the heterogeneity between pipe and PS-Path increases, the improvement brought by MPBond becomes larger.

6.4 Varying Network Conditions

We next evaluate how MPBond performs under changing network conditions. We focus on the 2-device case where Pixel 2 is the primary and Nexus 6P is the helper. We first replay the real WWAN and WLAN bandwidth profiles we collected in §3. Figure 13 shows the download time of different schemes. Compared to kibbutz (COMBINE), MPBond reduces the download time by 21%-23% (29%-35%). The corresponding energy consumption reduction is 18%-25% (16%-23%), as shown in Figure 14. This again shows that leveraging helper-side connection split, buffer management, and the judiciously designed PAMS scheduler helps MPBond to achieve high network utilization under fluctuating network conditions.

In-the-wild Experiments. To further understand the benefits of MPBond, we conduct field test in real world settings. We focus on comparing MPBond with kibbutz whose performance is closer to MPBond. Specifically, we conduct experiments at two locations by performing 1-min download for each scheme back-to-back at each place and repeat it for 10 times. We measure the instantaneous throughput every 100ms. Figure 15 shows the throughput distribution of MPBond and kibbutz. At the first location, MPBond improves the median throughput by 13% compared to kibbutz. At the second one, the improvement is 23%. MPBond also greatly reduces the low throughput periods, with a 90% improvement of 5th percentile throughput over kibbutz in both locations, due to its buffer management and helper-side connection split that exploit the capacity of the fluctuating WWAN and WLAN links as much

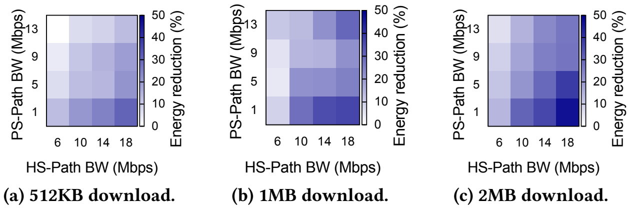


Figure 11: **Energy consumption reduction: MPBond compared to kibbutz.**

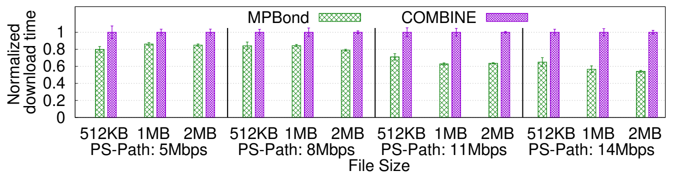


Figure 12: **Performance of MPBond v.s. COMBINE under different BW combinations: PS-Path: {5, 8, 11, 14} Mbps, HS-Path: 10Mbps, pipe: 5Mbps.**

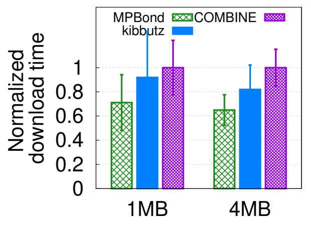


Figure 13: **Download time under varying network condition.**

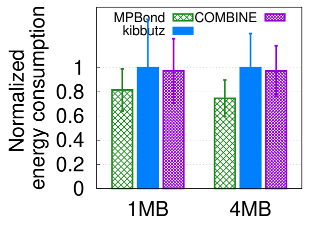


Figure 14: **Energy consumption under varying network condition.**

as possible (§4.2). The energy per byte is improved by 19% and 24% at the two places respectively (not shown in the figure).

6.5 Video Streaming Performance

All experiments so far use bulk file download as the workload. We now examine how MPBond helps improve the QoE and energy efficiency of video streaming, one of the applications that dominate mobile network traffic. We stream adaptive bitrate (ABR) videos using Exoplayer [9] to study the impact of different schemes on video bitrate. We use three video settings: Big Buck Bunny with 2-second segment duration (B2), Tears of Steel with 2-second segment duration (T2), and Tears of Steel with 6-second segment duration (T6). Big Buck Bunny has 20 bitrates ranging from 46kbps to 4.2Mbps, while Tears of Steel has 9 bitrates ranging from 253kbps to 10Mbps. The total video duration for them are 596s and 734s, respectively.

We focus on the comparison between MPBond and kibbutz, both of which don't require modification to the video streaming application. Figure 16 shows the video bitrate and per device energy consumption. With two devices, MPBond reduces the energy consumption by 13%-14% compared to kibbutz, while achieving similar video bitrate. When the number of devices become three, MPBond improves the video bitrate by 118% compare to kibbutz for two (T2 and T6) of the three settings. The rest one (B2) doesn't show much improvement since using two devices can already reach the highest bitrate. Nevertheless, the per device energy consumption in B2 is reduced because of the help of the third device. We further make two observations in T2 and T6: (1) MPBond-Naive achieves even lower energy consumption compared to MPBond and kibbutz, but with the cost of a lower video bitrate. (2) While MPBond helps improve the bitrate as the number of devices increases, the per device energy consumption doesn't get reduced like bulk download does, because a higher bitrate corresponds to a larger video segment: this is a classic tradeoff between QoE and data usage in ABR streaming.

360-degree Video Streaming. 360 degree video streaming has a much higher bandwidth requirement compared to regular video streaming and is an ideal use case for MPBond. For our experiment, the video bitrate is fixed at 64 Mbps. Since the mobile devices we have do not support the decoding of such high definition, we instead employ a video player emulator to download the video content without rendering it. We employ video stall ratio (stall time divided by video length) as the QoE metric and vary the number of device from 1 to 3 to study how much improvement MPBond brings. For the single primary device the stall ratio is as high as 145%, while using a helper helps reduce it to 27%. It's further reduced to 3% when three devices are used – this clearly shows that in reality the fluctuating LTE oftentimes does not always meet the high bandwidth requirement of 360-degree videos and further motivates the need of MPBond to support more than 2 devices.

6.6 Leveraging the Dual Mode

We now evaluate the benefits of dual mode by involving two users, each carrying a smartphone (Pixel 2/Nexus 6P) with LTE connectivity (T-Mobile/AT&T). Both of their LTE are capped to 5Mbps. The pipe is unthrottled. To examine how much improvement MPBond's dual mode brings, the two users start the following workload at the same time: sequential 1MB chunks are requested on each smartphone, with the inter-chunk time being a random number between 1 and 5 seconds, emulating the video streaming traffic. Figure 8 compares the chunk download time in dual mode of MPBond and the download time when each of them download independently without MPBond. As shown, the download time is improved by 32% and 29% for Pixel 2 and Nexus 6P, respectively.

6.7 Indoor Applicability

Above experiments focus on MPBond's main use case – outdoor cellular networking. Now we consider indoor environments where WiFi infrastructures most likely exist. When there's a WiFi infrastructure, MPTCP over WiFi and LTE can be easily applied for bandwidth aggregation. To understand how MPBond compares to it, we conduct 4MB file download experiments at two different indoor locations to study the performance and cellular data usage of two schemes: (1) MPBond on a primary and a helper where the PS-Path and the HS-Path are over LTE, pipe is over WiFi, (2) MPTCP over WiFi and LTE on the primary. The two locations are with different WiFi network conditions: location A has a high average WiFi signal strength of -51dBm while location B receives weaker WiFi signals (-68dBm on average). Across 10 back-to-back runs, the average file download times of scheme 1 and 2 at location A are 1.5s and 1.0s, respectively. At location B, the corresponding download times for scheme 1 and 2 are 1.2s and 1.6s, respectively. The cellular data usage of scheme 2 at location A and B are 1.6MB and

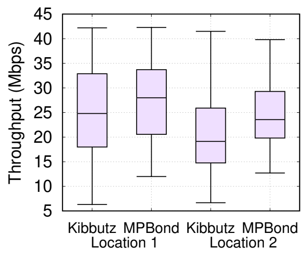


Figure 15: Results of in-the-wild experiments.

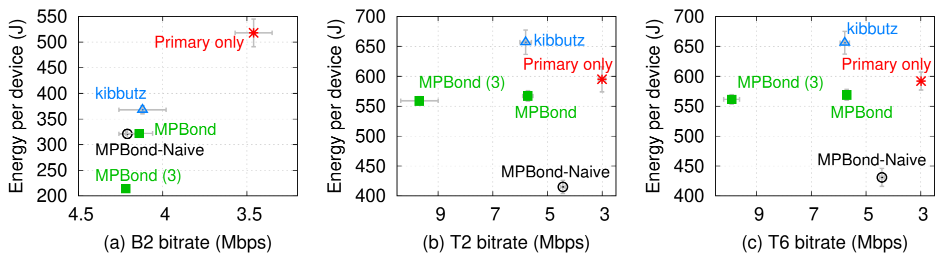


Figure 16: Video streaming QoE & energy. (PS-Path: 5Mbps, HS-Path: 10Mbps, pipe: 5Mbps): Single device (Pixel2), MPBond (Pixel2+Nexus6P), MPBond w/ 3 devices (Pixel2+Nexus6P+LG2), and kibbutz (Pixel2+Nexus6P).

3.3MB, respectively. The results show that (1) MPBond always has a higher cellular data (metered) usage (4MB) compared to scheme 2, (2) depending on the WiFi network condition, MPBond may either outperform (*e.g.,* at location B) or underperform (*e.g.,* at location A) scheme 2. In indoor environments with good WiFi networks such as location A, an MPBond user can choose to fall back to scheme 2, the regular multipath over WiFi and LTE, *e.g.,* by leveraging context information [44]. We leave developing a full-fledged context-aware framework for automatic switching between scheme 1 and 2 as future work.

6.8 System Overhead and Energy Concerns

We measure the CPU utilization on the MPBond primary as well as the helper when running the same workload as kibbutz: downloading a large file from remote server. We repeat the experiment for 10 times and the average extra CPU utilization compared to kibbutz is no more than 4% for both the primary and the helper. We also answer the question left in §3.3 to examine the battery drain of a wearable when acting as a helper. We stream a 15-min video and examine the battery drain of a fully charged LG Urbane 2 smartwatch. We repeat it for 10 times and observed no more than 7% average battery drop: this shows the feasibility of our solution.

The previous energy measurement results in the evaluation section are based on power models instead of hardware tools. To understand the measurement errors (*i.e.,* model inaccuracies), we now use a commercial power monitor [10] to measure the real energy consumption. We employ a Samsung Galaxy S5 smartphone that can to be hooked by the power monitor. We use it as the helper and a Pixel 2 as the primary. Our workload is downloading a 4MB file under the following network setting: PS-Path: 8Mbps, HS-Path: 10Mbps, pipe: 5Mbps. The power monitor measures the helper-side energy consumption with both MPBond and COMBINE. We focus on energy measurement on the helper due to the limited number of power monitors we have and a helper is usually less powerful and more energy-constrained (*e.g.,* a wearable). We repeat the experiment 10 times and the helper on average consumes 2.3J and 3.4J energy for MPBond and COMBINE, respectively. Compared to the absolute energy numbers derived from models (1.9J and 2.6J), the error can be as high as 24%. However, the difference between model-based (27%) and power monitor-based (32%) relative energy reductions (MPBond over COMBINE) is as low as 5%.

7 DISCUSSION

MPBond over Homogeneous Cellular Links. The experiments in §6 are conducted with mobile devices connecting to different ISPs, providing MPBond with ideal opportunities for bandwidth aggregation. We now examine another case where an MPBond primary (Pixel 2) and the helper (Nexus 6P) use the same cellular carrier (AT&T). We perform 1-min downloads at a residential apartment using this setup and compare it with a single device setup (Nexus 6P). The experiments are conducted back to back and repeated for 10 times. The average throughput of using the Nexus 6P alone is measured to be 10.1Mbps. Leveraging MPBond improves the average throughput to be 15.1 Mbps, with 8.2 Mbps coming from the direct subflow and 6.9 Mbps from the indirect one. As we can see, even if the throughput of the primary subflow decreases with the addition of a helper device, there is a still fair amount of gain on the overall throughput. This is because the eNodeB does not always allocate all the resource blocks to a single UE, due to the scheduling policy and the existence of other single-device users. As a result, the MPBond user still gets a higher overall throughput compared to using her one device.

MPBond over Congested Networks. The above indicates that MPBond users typically get a higher share of the wireless resources compared to single-device users, with devices connecting to either a single carrier or multiple carriers. Now we discuss what happens when people all use MPBond, especially when the wireless spectrum is crowded during peak hours. In this zero-sum situation, each MPBond user cannot gain extra bandwidth share at the same time. However, we expect its impact on MPBond's use is limited given the bursty nature of today's Internet traffic. Besides, an MPBond user can configure the number of helper devices to get a desired share among other MPBond users with her affordable cost. Note that this is a generic problem of mobile multipath and multi-device network-level collaboration.

8 CONCLUSION

MPBond is an efficient network-level collaboration framework for personal mobile devices. We develop mechanisms for connection and buffer management, packet scheduling, and policy enforcement and demonstrate the performance and energy benefits of MPBond using real-world mobile devices, applications, and networks.

ACKNOWLEDGEMENTS

We would like to thank our shepherd, Aruna Balasubramanian, and the anonymous reviewers for their valuable comments. This work was partially supported by NSF under the grants CNS-1827940, CCF-1628991, CNS-1629763, CNS-1917424, and CCF-1903880.

REFERENCES

[1] 2014. People for Whom One Cellphone Isn't Enough. https://www.wsj.com/articles/people-who-use-two-cellphones-1396393393.
[2] 2015. Doing the Two-Smartphone Shuffle. https://geekdad.com/2015/02/two-smartphone-shuffle/.
[3] 2015. How Often Does the Average American Replace His or Her Smartphone? https://www.fool.com/investing/general/2015/07/15/how-often-does-the-average-american-replace-his-or.aspx.
[4] 2016. 3 Reasons Why You Should Own A Second Cell Phone. https://www.forbes.com/sites/forbesmarketplace/2016/03/17/3-reasons-why-you-should-own-a-second-cell-phone/.
[5] 2017. 8 Frugal Reasons to Have Two Phones. https://www.thefrugalgene.com/frugal-phones/.
[6] 2017. "Multiple phone personality" is trending. https://hackernoon.com/multiple-phone-personality-is-trending-2c1670bd7367.
[7] 2018. Cisco Visual Networking Index: Forecast and Trends, 2017-2022 White Paper. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.
[8] 2018. Using Your Old Smartphone as a Mobile Hotspot. https://www.hellotech.com/blog/using-old-smartphone-as-mobile-hotspot/.
[9] 2019. Exoplayer. https://google.github.io/ExoPlayer.
[10] 2019. Monsoon Power Monitor. https://www.msoon.com/online-store.
[11] 2020. *MPBond* github repository. https://github.com/XiaoShawnZhu/MPBond.
[12] Ardalan Amiri Sani, Kevin Boos, Min Hong Yun, and Lin Zhong. 2014. Rio: a system solution for sharing i/o between mobile systems. In *MobiSys*. ACM.
[13] Ganesh Ananthanarayanan, Venkata N Padmanabhan, Lenin Ravindranath, and Chandramohan A Thekkath. 2007. Combine: leveraging the power of wireless peers through collaborative downloading. In *MobiSys*. ACM.
[14] Hari Balakrishnan, Hariharan S Rahul, and Srinivasan Seshan. 1999. An integrated congestion management architecture for Internet hosts. *ACM SIGCOMM Computer Communication Review* 29, 4 (1999), 175–187.
[15] Xiaomeng Chen, Ning Ding, Abhilash Jindal, Y Charlie Hu, Maruti Gupta, and Rath Vannithamby. 2015. Smartphone energy drain in the wild: Analysis and implications. *ACM SIGMETRICS Performance Evaluation Review* 43, 1, 151–164.
[16] Xavier Corbillon, Ramon Aparicio-Pardo, Nicolas Kuhn, Géraldine Texier, and Gwendal Simon. 2016. Cross-layer scheduler for video streaming over MPTCP. In *MMSys*. ACM.
[17] Andrei Croitoru, Dragos Niculescu, and Costin Raiciu. 2015. Towards Wifi Mobility without Fast Handover.. In *NSDI*. USENIX.
[18] Quentin De Coninck and Olivier Bonaventure. 2017. Multipath quic: Design and evaluation. In *CoNEXT*. ACM.
[19] Quentin De Coninck and Olivier Bonaventure. 2018. Tuning multipath TCP for interactive applications on smartphones. *IFIP Networking 2018* (2018).
[20] Shuo Deng, Ravi Netravali, Anirudh Sivaraman, and Hari Balakrishnan. 2014. Wifi, lte, or both?: Measuring multi-homed wireless internet performance. In *IMC*. ACM, 181–194.
[21] Alexander Frommgen, Tobias Erbshäußer, Alejandro Buchmann, Torsten Zimmermann, and Klaus Wehrle. 2016. Remp tcp: Low latency multipath tcp. In *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 1–7.
[22] Yihua Ethan Guo, Ashkan Nikravesh, Z Morley Mao, Feng Qian, and Subhabrata Sen. 2017. Accelerating multipath transport through balanced subflow completion. In *MobiCom*. ACM.
[23] Bo Han, Feng Qian, Shuai Hao, and Lusheng Ji. 2015. An anatomy of mobile web performance over multipath TCP. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. ACM, 5.
[24] Bo Han, Feng Qian, Lusheng Ji, and Vijay Gopalakrishnan. 2016. MP-DASH: Adaptive video streaming over preference-aware multipath. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. ACM, 129–143.
[25] Junxian Huang, Feng Qian, Yihua Guo, Yuanyuan Zhou, Qiang Xu, Z Morley Mao, Subhabrata Sen, and Oliver Spatscheck. 2013. An in-depth study of LTE: effect of network protocol and application behavior on performance. In *SIGCOMM*. ACM.
[26] Lorenzo Keller, Anh Le, Blerim Cici, Hulya Seferoglu, Christina Fragouli, and Athina Markopoulou. 2012. Microcast: Cooperative video streaming on smartphones. In *MobiSys*. ACM.
[27] Kyu-Han Kim and Kang G Shin. 2005. Improving TCP performance over wireless networks with collaborative multi-homed mobile hosts. In *MobiSys*. ACM.
[28] Zeqi Lai, Y Charlie Hu, Yong Cui, Linhui Sun, and Ningwei Dai. 2017. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. In *MobiCom*. ACM.
[29] HyunJong Lee, Jason Flinn, and Basavaraj Tonshal. 2018. RAVEN: Improving Interactive Latency for the Connected Car. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM.
[30] Li Li, Ke Xu, Tong Li, Kai Zheng, Chunyi Peng, Dan Wang, Xiangxiang Wang, Meng Shen, and Rashid Mijumbi. 2018. A measurement study on multi-path tcp with multiple cellular carriers on high speed rails. In *SIGCOMM*. ACM, 161–175.
[31] Yeon-sup Lim, Erich M Nahum, Don Towsley, and Richard J Gibbens. 2017. Ecf: An mptcp path scheduler to manage heterogeneous paths. In *CoNEXT*. ACM.
[32] Xing Liu, Tianyu Chen, Feng Qian, Zhixiu Guo, Felix Xiaozhu Lin, Xiaofeng Wang, and Kai Chen. 2017. Characterizing smartwatch usage in the wild. In *MobiSys*. ACM, 385–398.
[33] Yuanwei Lu, Guo Chen, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, Enhong Chen, and Thomas Moscibroda. 2018. Multi-path transport for RDMA in datacenters. In *NSDI*. USENIX.
[34] Arvind Narayanan, Eman Ramadan, Jason Carpenter, Qingxu Liu, Yu Liu, Feng Qian, and Zhi-Li Zhang. 2020. A First Look at Commercial 5G Performance on Smartphones. In *Proceedings of The Web Conference 2020*.
[35] Cătălin Nicutar, Dragoş Niculescu, and Costin Raiciu. 2014. Using cooperation for low power low latency cellular connectivity. In *CoNEXT*. ACM, 337–348.
[36] Ashkan Nikravesh, Yihua Guo, Feng Qian, Z Morley Mao, and Subhabrata Sen. 2016. An in-depth understanding of multipath TCP on mobile devices: Measurement and system design. In *MobiCom*. ACM.
[37] Ashkan Nikravesh, Yihua Guo, Xiao Zhu, Feng Qian, and Z Morley Mao. 2019. MP-H2: A Client-only Multipath Solution for HTTP/2. In *MobiCom*. ACM.
[38] Sangeun Oh, Ahyeon Kim, Sunjae Lee, Kilho Lee, Dae R Jeong, Steven Y Ko, and Insik Shin. 2019. FLUID: Multi-device Mobile Platform for Flexible User Interface Distribution. In *MobiCom*. ACM.
[39] Sangeun Oh, Hyuck Yoo, Dae R Jeong, Duc Hoang Bui, and Insik Shin. 2017. Mobile plus: Multi-device mobile platform for cross-device functionality sharing. In *MobiSys*. ACM.
[40] Christoph Paasch, Simone Ferlin, Ozgu Alay, and Olivier Bonaventure. 2014. Experimental evaluation of multipath TCP schedulers. In *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*. ACM, 27–32.
[41] Feng Qian, Vijay Gopalakrishnan, Emir Halepovic, Subhabrata Sen, and Oliver Spatscheck. 2015. TM 3: flexible transport-layer multi-pipe multiplexing middlebox without head-of-line blocking. In *CoNEXT*. ACM.
[42] Feng Qian, Bo Han, Jarrell Pair, and Vijay Gopalakrishnan. 2019. Toward Practical Volumetric Video Streaming on Commodity Smartphones. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*. ACM.
[43] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Zhuoqing Mao, Subhabrata Sen, and Oliver Spatscheck. 2011. Profiling resource usage for mobile applications: a cross-layer approach. In *MobiSys*. ACM.
[44] Valentin Radu, Panagiota Katsikouli, Rik Sarkar, and Mahesh K Marina. 2014. A semi-supervised learning approach for robust indoor-outdoor detection with smartphones. In *SenSys*. ACM.
[45] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. 2011. Improving Datacenter Performance and Robustness with Multipath TCP. In *ACM SIGCOMM*.
[46] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. 2012. How hard can it be? designing and implementing a deployable multipath TCP. In *NSDI*. USENIX.
[47] Swetank Kumar Saha, Shivang Aggarwal, Rohan Pathak, Dimitrios Koutsonikolas, and Joerg Widmer. 2019. MuSher: An Agile Multipath-TCP Scheduler for Dual-Band 802.11 ad/ac Wireless LANs. In *MobiCom*. ACM.
[48] Ashish Sharma, Vishnu Navda, Ramachandran Ramjee, Venkata N Padmanabhan, and Elizabeth M Belding. 2009. Cool-tether: energy efficient on-the-fly wifi hot-spots using mobile phones. In *CoNEXT*. ACM.
[49] Hang Shi, Yong Cui, Xin Wang, Yuming Hu, Minglong Dai, Fanzhao Wang, and Kai Zheng. 2018. STMS: Improving MPTCP Throughput Under Heterogeneous Networks. In *USENIX ATC*. 719–730.
[50] Varun Singh, Saba Ahsan, and Jörg Ott. 2013. MPRTP: multipath considerations for real-time media. In *Proceedings of the 4th ACM Multimedia Systems Conference*. ACM, 190–201.
[51] Xing Xu, Yurong Jiang, Tobias Flach, Ethan Katz-Bassett, David Choffnes, and Ramesh Govindan. 2015. Investigating transparent web proxies in cellular networks. In *International Conference on Passive and Active Network Measurement*. Springer, 262–276.
[52] Xiao Zhu, Yihua Ethan Guo, Ashkan Nikravesh, Feng Qian, and Z Morley Mao. 2019. Understanding the Networking Performance of Wear OS. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3, 1 (2019), 3.