Rethink Phone-Wearable Collaboration From the Networking Perspective

Xing Liu*
Indiana University
Bloomington, IN
xl45@indiana.edu

Yunsheng Yao*
Indiana University
Bloomington, IN
yy29@indiana.edu

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

ABSTRACT

We routinely carry wearables such as smartwatches and smart glasses with our smartphones. In this paper, we explore the feasibility of leveraging wearables' network interfaces to provide extra benefits such as enhanced network performance and better security for smartphones. To provide the protocol support for these new use cases, we propose a novel multipath scheme called Distributed Mobile Multipath (DMM), which generalizes the multipath support to multiple hosts by allowing subflows to be established from different mobile devices. To demonstrate the potential benefits of DMM, we conduct preliminary field experiments, which show that in public locations with free WiFi, leveraging the smartwatch's WiFi interface can increase the smartphone's download throughput by 70%.

CCS Concepts

•**Networks** → **Network protocol design;** •**Human-centered computing** → **Ubiquitous and mobile computing;**

Keywords

Distributed mobile multipath transport (DMM); Multipath TCP; Wearable devices; Phone-wearable collaboration

1. INTRODUCTION

Many wearable devices such as smartwatches and smart glasses need to communicate with the external world. This is usually achieved by two ways: (1) pairing with a smartphone and using it as an Internet gateway, or (2) establishing standalone Internet connections using wearables' built-in WiFi or cellular interface. Modern wearables often have standalone network interfaces. For example, most medium-end smartwatches have WiFi, and some high-end watches such as LG Watch Urbane 2 are even equipped with LTE.

These two communication paradigms enable a wide range of application use cases. In particular, the phone plays an important role in assisting a wearable in getting the Internet access. However, on the opposite side, it is oftentimes difficult for a wearable to assist the phone due to a lack of support from the networking protocol stack. Let us consider the following scenario: Alice stays in a hotel that offers free WiFi with its rate being limited to 1Mbps per device. Meanwhile she wants to watch a video on her smartphone whose bitrate is 1.6Mbps, which is clearly not achievable without stalls through the free WiFi using a single mobile device. Nevertheless, Alice also wears a smartwatch with a WiFi interface. Ideally, if the watch and phone stream the video *jointly*, Alice would be able to watch the video smoothly without purchasing the premium WiFi plan or using her limited cellular data. Through pilot experiments, we demonstrate in §4 that the above scenario is prevalent in real-world locations such as restaurants, shopping centers, and buses, where *per-device rate limiting* is often enforced. Given the co-presence of smartphones and wearables, at these places, a user can conveniently leverage her wearable to accelerate the data transfer for her phone.

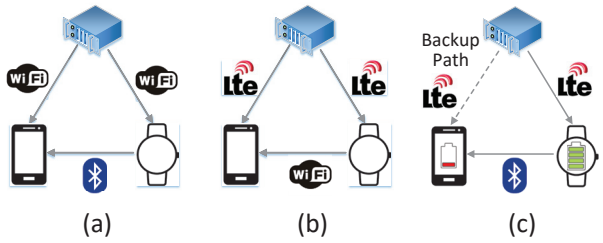

Figure 1: Boost network performance for the primary device (*e.g.,* a smartphone) using a secondary device (*e.g.,* a smartwatch).

1.1 Approach Overview

Inspired by the above use case, the high-level idea of our proposal is intuitive: *as we routinely carry both wearables and phones, their network interfaces can be leveraged in a cooperative manner to enhance the phone-side network performance, in an application-transparent manner.* This is illustrated in Figure 1(a). The phone and wearable download different portions of the same content from the server through their separate WiFi interfaces. Upon receiving the data, the wearable sends it to the phone using a local link such as Bluetooth or Direct WiFi. The phone then merges the portion obtained from the wearable with its own downloaded portion, and delivers the reassembled data stream to the application. Note that this paradigm can be configured in different ways. As shown in Figure 1(b), if both devices have a cellular interface, the two cellular paths can also be combined

*Co-primary student authors.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

WearSys'17, June 19, 2017, Niagara Falls, NY, USA
© 2017 ACM. ISBN 978-1-4503-4959-8/17/06…$15.00
DOI: http://dx.doi.org/10.1145/3089351.3089356

to enhance the performance. Figure 1(c) shows a scenario where the phone "onloads" the Internet connection to the wearable. This sounds counter-intuitive but we do believe there are valid use cases. For example, in the hotel room, Alice can leverage the watch as an LTE/WiFi range extender by placing it at a spot (*e.g.,* by the window) with good signal strength. In another example, when the battery level of the phone is critical, it also makes sense to onload cellular connectivity, which is power-hungry [4], to the wearable (with sufficiently high battery level). One issue here is the potentially high energy consumption of the wearable. This is indeed a valid concern. However we argue that in many scenarios, users do have incentives to trade an acceptable amount of wearable's energy for better performance, as to be further justified in §5.

Now consider how to realize the schemes in Figure 1. They can be realized at the application layer (*e.g.,* using separate HTTP byte range requests) as done by prior cross-device data sharing systems [9, 1, 17]. Despite its simplicity, this approach lacks generality. We instead advocate for *a transport-layer OS service for phone and wearable to cooperatively fetch contents with application transparency.*

As readers may know, if the two interfaces are on the *same* host, their cooperation can be achieved by multipath, a popular system solution allowing applications to simultaneously use more than one network paths. The *de-facto* multipath solution is Multipath TCP (MPTCP [5]), which transparently stripes a user TCP connection's data across multiple network paths (*subflows*) by adding a shim layer to TCP. MPTCP has registered wide deployment.

In our scenario, since the two subflows reside on different hosts, directly applying MPTCP is difficult. We thus propose a transport-layer cross-device connection sharing scheme called Distributed Mobile Multipath Transport (DMM). As its name suggests, the basic idea of DMM is to generalize the multipath support to multiple mobile hosts by allowing the subflows to be established from different client hosts. In Figure 1(a), the two subflows are phone–server and watch–server. When the watch receives the data, it will then forward the data over a custom data channel to the phone.

In §2, we detail the design of DMM. DMM's capability goes beyond just speeding up network transfers. In §3, we show another use case of DMM for protecting users' sensitive data on smartphones. In §4, we present preliminary results showing that in public locations with free WiFi, leveraging a smartwatch's WiFi interface can increase the smartphone's download throughput by 70% (the median value). We discuss several practical issues in §5, summarize related work in §6, before concluding the paper in §7.

1.2 Contributions

Overall, this paper makes the following contributions.

- Given the co-presence of wearables and smartphones in our daily life, we propose novel use cases of leveraging wearables to improve the network performance, energy efficiency, and security for the smartphone.
- We propose DMM, a lightweight cross-device connection sharing scheme that serves as the underlying protocol for the above use cases. Note that DMM is a general extension of the multipath transport and its applicability goes beyond wearable and mobile networking.
- We conduct preliminary field studies to demonstrate the feasibility of leveraging wearables to boost the network performance for the smartphone.

2. DMM: DISTRIBUTED MOBILE MULTIPATH TRANSPORT

DMM is motivated by the increasingly prevalent deployment of multipath solutions such as MPTCP. As its name suggests, the basic idea of DMM is to generalize multipath to multiple mobile hosts by allowing the subflows to be established from different client hosts. In Figure 1(a), the two subflows are phone–server and watch–server, which belong to the same TCP connection established by the application. When the watch receives the data, it will then forward the data over a custom data channel, which we call a *pipe*, to the phone. The pipe is transparent to the server, which only sees the two subflows as a normal multipath server would perceive.

2.1 Why Not the Tethering Approach?

Before describing our proposal, we first consider an alternative approach that leverages *tethering*. After being tethered to the phone (*e.g.,* via USB or WiFi), a wearable will essentially appear as a virtual network interface on the phone. The phone can then apply off-the-shelf MPTCP to both its local interface and the virtual interface. This approach is used by some existing system such as Mobile kibbutz [12]. Compared to the tethering approach, DMM offers several benefits and new features as highlighted below.

- In the tethering approach, the wearable acts as a simple layer-3 router that blindly forwards the data. DMM instead works at Layer 4. It allows various types of transport-layer enhancement and policies to be carried out over the pipe, leading to better performance and reliability (§2.3).
- Oftentimes more than one link can potentially be used for tethering (*e.g.,* WiFi, Bluetooth, and Bluetooth Low Energy) and it is up to the user to make a static configuration. DMM instead allows the pipe to be dynamically selected based on the subflow data rate to balance the tradeoff between the performance and the wearable's energy consumption.
- In the tethering approach, when the phone-wearable connectivity is lost (*e.g.,* when the wearable temporarily moves out of the Bluetooth range), the virtual interface will go down and all its associated states will be lost. DMM, instead, can maintain the transport-layer states (*e.g.,* packets buffered at the wearable) to facilitate the recovery of the subflow when the phone-wearable connectivity comes back.
- DMM can be extended to enable more sophisticated use cases that cannot be supported by tethering (§3).

We next present the design of DMM, which extends MPTCP, the state-of-the-art multipath solution. For the purpose of applying DMM in the wearable context, we assume that all distributed mobile devices are trusted. Also, in this paper we limit our discussions to two subflows (phone-server and wearable-server) despite our confidence of extending DMM to multiple subflows.

2.2 The Basic Design

TCP ensures reliable process-to-process communication. Therefore, in DMM, although a TCP connection spans across devices, at any given time only two user processes are involved in sending and receiving application data. We call the client device on which the user process resides the *primary* device and the other device the *secondary* device. Without loss of generality, let the primary device be the phone and the secondary device be the wearable. We assume that the data channel between the phone and the wearable has already been established (*e.g.,* the wearable has paired with the phone). Recall that such a channel is called a *pipe*.

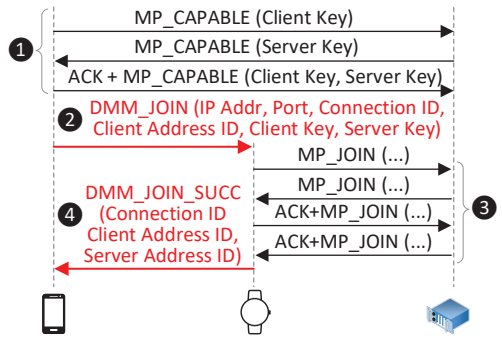

Figure 2: DMM connection handshake.

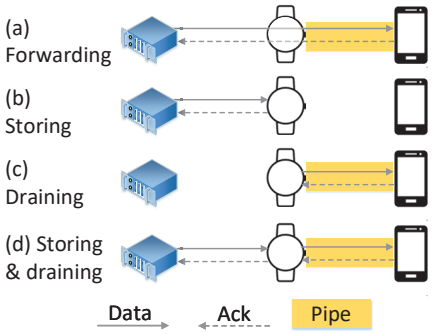

Figure 3: Different working modes of DMM.

Connection Establishment. Figure 2 describes how a DMM-capable MPTCP connection is established. While the high-level procedure is somewhat similar to the MPTCP connection establishment, the two devices need to exchange some information over the pipe. Specifically, Step ❶ and ❸ perform handshakes for the phone's subflow and the wearable's subflow, respectively (same as those in MPTCP if the subflows are on the same host); Step ❷ and ❹ are new control messages being exchanged over the pipe. In Step ❷, the phone informs the wearable of necessary information: the server IP address and port number tell the wearable which server to connect; the client/server keys exchanged in the phone's subflow handshake will be used to craft the `MP_JOIN` messages so that the server can authenticate the wearable's subflow and associate it with the phone's; the client `Address ID` will also appear in an `MP_JOIN` message as a reference to the subflow's address; the `Connection ID` is a new identifier introduced by DMM to refer to the whole connection when data is exchanged over the pipe, over which multiple connections are multiplexed. In Step ❹, the phone is notified when the wearable's subflow is established. Besides connection establishment, procedures for subflow management, connection shutdown, and error handling can also be designed, with details omitted here.

Data Transfer. For download, the phone and the wearable separately maintain subflow-level receive buffers. In addition, the phone (the primary device) also needs to maintain the meta receive buffer where all subflows' data is reassembled into the original byte stream. When the wearable receives any data, the data (with its connection-level sequence number and `Connection ID` attached) will be forwarded over the pipe to the phone. For data upload, the phone maintains the meta send buffer. It runs a scheduling algorithm (*e.g.,* MinRTT[1] or round robin) that stripes the application data stream onto the two subflows. For the wearable's subflow, the data first needs to be forwarded to the wearable and then be transmitted to the server. Note that during download (upload), when the wearable receives any data from the server (phone), it should not ACK the data by itself. Instead, it should wait for the ACK from the phone (server) and forward the ACK to the server (phone). Doing so offers two advantages. First, it allows the sender to accurately estimate the end-to-end RTT and thus to make correct scheduling decisions (*e.g.,* for MinRTT). Second, it also provides better fault tolerance by allowing the server to retransmit the data when the pipe is lost.

Server Transparency. It is important to note that DMM is fully transparent to the server, whose MPTCP stack still thinks the two subflows originate from the same host. In this way, DMM can immediately work with existing servers and proxies that support MPTCP.

[1]MinRTT is the default scheduler used by MPTCP. It always picks the path with the smallest RTT to transmit a packet.

2.3 Pipe Realization and Management

A key primitive introduced by DMM is the pipe. It provides a flexible layer-4 channel to bridge the phone and the wearable. A pipe can be realized by various types of radio links such as regular WiFi, direct WiFi, classical Bluetooth (BT), and Bluetooth Low Energy (BLE). If the phone and wearable are in the same local 802.11 network, they can communicate via regular WiFi. Otherwise direct WiFi can be used to establish a device-to-device (D2D) link as the pipe. Many of today's WiFi chips (*e.g.,* Broadcom BCM4325 and Qualcomm WCN1312) allow the device to simultaneously connect to a regular WiFi (used for a subflow) and WiFi direct (used for the pipe). The pipe can also be realized by BT or BLE that are more energy-efficient.

Dynamic Pipe Selection. When multiple wireless links are available for realizing the pipe, DMM needs to select one judiciously. On one hand, the bandwidth of the pipe needs to be higher than that of the wearable-server path to prevent the pipe from becoming the performance bottleneck. On the other hand, using a pipe whose bandwidth is too high may waste energy. We plan to design an algorithm that dynamically selects the pipe's link based on the measured wearable-server path bandwidth, the power consumption profile of the candidate wireless links, and the application requirement for throughput and latency. A change of the network condition may result in a change of the pipe's link.

Storing and Draining Mode. In our basic design (§2.2), the watch simply forwards the data and ACKs it receives. We call this the *forwarding mode* as shown in Figure 3(a). DMM also introduces two other modes illustrated in Figure 3(b) and 3(c): the *storing mode* and the *draining mode*. In the storing mode, the wearable receives and ACKs the data from the server by itself. The data will then be stored (buffered) locally without being sent to the phone. On the phone side, since the wearable subflow's data is missing, the phone also has to buffer the data received from its own subflow. Since the server maintains the mappings between subflow-level sequence numbers and connection-level sequence numbers, it can tell which portion of the file has been received (by either the wearable or the phone) from each subflow's ACK stream. In the draining mode, the wearable sends over the pipe its buffered data to the phone, which can then perform reassembly and deliver the data to the application. It is important to note that (1) the storing and draining mode can happen in parallel when the wearable-server path has a higher throughput than the pipe, as shown in Figure 3(d); (2) these two modes should only be used for downloading delay-

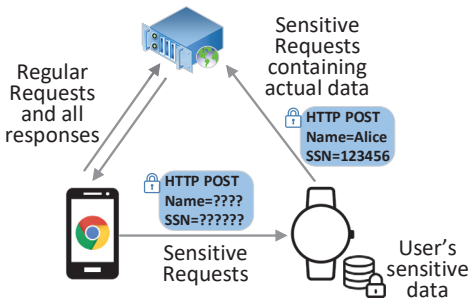

Figure 4: Wearable-assisted SUD protection.

tolerant contents because the storing mode will delay the initial delivery of the content to the application; (3) if the wearable with stored data is lost permanently (*e.g.,* running out of battery), the entire connection needs to be aborted.

There are several use cases of the storing and draining modes. For example, consider again Figure 1(a). Suppose that due to hardware and network restriction, on the wearable side, when WiFi is used by the WAN (*i.e.,* the Internet), the only available link for the pipe is BT whose bandwidth is lower than WiFi. In this case, the data rate of the wearable's subflow will be limited by the BT link. If the data transfer is delay-tolerant, the wearable can first switch to the storing mode and meanwhile perform the (optional) draining over BT. When the wearable has received all data, it will change the pipe from BT to WiFi over which the remaining stored data will be drained to the phone at a high speed. Such a "store-then-drain" scheme reduces the overall download time at the cost of delaying the initial file delivery.

Handling Fluctuating Pipe Connectivity. Another use case of the storing mode is to deal with temporarily poor or lost pipe connectivity. This may happen when the wearable is in mobility. In this case (assuming a download is in progress), the wearable can enter the storing and draining mode to prevent the pipe from slowing down the WAN-side download. When the pipe connectivity becomes stable, the wearable will then switch back to the normal forwarding mode. The wearable can perform early detection of the poor pipe connectivity using, for example, the decrease of the pipe's available bandwidth or signal strength as indicators.

3. DMM USE CASE: PROTECT SENSITIVE DATA USING WEARABLE

Besides improving the performance, DMM can be flexibly used for other purposes. Here we exemplify a use case of using wearables to enhance smartphone security.

It is common that users store various types of sensitive user data (SUD) such as credit card numbers and social security numbers on their smartphones. Unfortunately, smartphones are vulnerable to a wide range of attacks that may cause such sensitive data to be leaked. For example, the phone can be physically stolen or get unauthorized access; stealthy spyware and malware may discover and leak confidential information in various ways.

We propose to leverage the wearable to protect the SUD. The basic idea is illustrated in Figure 4. The SUD is stored on a wearable, which is assumed to be much more secure than a smartphone. We believe this is a valid assumption in most real-world scenarios: due to their wearable nature, wearables are less likely to be stolen or get unauthorized physical access; also their system software and applications are usually simpler, leading to a smaller number of attack vectors. As shown in Figure 4, regular requests and all responses are still sent/received by the phone. When the phone needs to send out any SUD, it composes a request (*e.g.,* an HTTPS POST request) as usual but with blank SUD fields and sends the request to the wearable. The wearable will then fill in the actual SUD and send the real request to the server. Note that the SUD is neither stored on nor transmitted by the phone[2]. Therefore even if the phone is stolen or compromised, the attacker cannot recover the SUD. Also the whole procedure is transparent to the server, which is not aware of the SUD being stored separately.

Within the above wearable-assisted SUD protection system, DMM plays an important role in ensuring that only the requests containing SUD, which usually account for a tiny portion within a persistent HTTPS session, are transmitted by the wearable. Otherwise (*e.g.,* if forwarding every request to the wearable), the incurred energy overhead on the wearable would be too high. Normally the phone as the primary device will only use its own subflow. For a request involving SUD, the phone will send the request with blank SUD to the wearable through the (encrypted) pipe. A special flag needs to be set so the wearable can perform further processing on the request before forwarding it to the server. The phone also needs to instruct DMM to swap the roles of the two devices: now the wearable becomes the primary device and the phone becomes the secondary device (recall in §2.2 that only the process on the primary device is involved in sending and receiving application data). This allows the SUD protection process on the wearable to perform the next step: fill the actual SUD into the request and transmit it over the wearable's subflow. After transmission, the wearable instructs DMM to swap the roles back: the phone now becomes the primary device and the wearable the secondary. Also, since the application TCP connection is encrypted by TLS (SSL), the TLS session key and the cipher algorithm need to be securely shared [19] between the phone and the wearable so that the wearable can encrypt the SUD request using the same session key before transmitting it. Note the initial TLS handshake that derives the session key can be solely performed on the phone.

Designing and implementing the entire system outlined above is a project of its own. Besides the key design decision of using DMM as the transport-layer infrastructure, we highlight several other design principles below. First, since HTTP(S) dominates the mobile application protocol usage, focusing on HTTPS request rewriting can satisfy most applications' requirements. Doing so makes the system lightweight compared to some existing approach that employs heavyweight solutions such as code offloading [19]. Second, we need to design a SUD management scheme allowing securely adding, removing, and modifying SUD. Third, to further enhance the security, we can strategically split a piece of SUD into two pieces that are stored on the phone and the wearable respectively. In this way, unless an attacker has access to *both* devices, the SUD will not be leaked. Each SUD can also be bound to a domain whitelist to prevent the SUD from being leaked to untrusted domains.

4. PRELIMINARY FIELD EXPERIMENTS

To assess the potential benefits of DMM for boosting network performance, we conduct preliminary field experiments at ten public places including hotels, airports, restaurants, shopping

[2]When the watch does not have the Internet access (but the phone does), the wearable can tunnel the request to the proxy (see §5) through the phone. The tunnel is established between the watch and the proxy. It is encrypted separately and independently so the phone is not able to access the tunneled data.

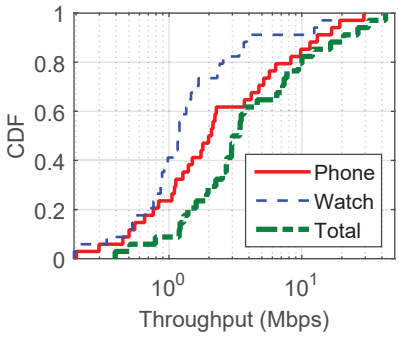

Figure 5: Distributions of phone-side, watch-side, and aggregated throughput.

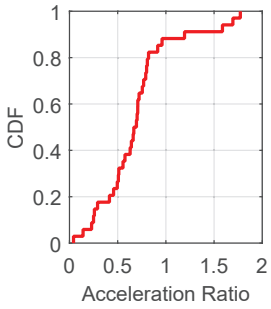

Figure 6: Distribution of acceleration ratio (watch-side throughput / phone-side throughput).

centers, and shuttle buses with free WiFi access. Our devices consist of a Nexus 5X smartphone (as the primary device) and an LG Urbane watch (as the secondary device). Both devices have a standalone WiFi interface. At each location, the phone and watch download a large file in parallel using the free WiFi, and their throughput is measured separately. As one device may finish the download earlier than the other, we only consider the period when the two devices are simultaneously receiving data. We use the sum of the throughput measured at each device to estimate how much speed-up DMM can bring by involving the wearable. Note that a limitation of our experiment is that we did not consider the pipe, which may throttle the wearable's subflow. We repeat the experiment at least three times at each location, yielding a total number of 36 measurements.

Figure 5 plots the distributions of the phone's average throughput, the watch's average throughput, and their sum across all experiments. We highlight several observations here. First, at most of these public locations, the free WiFi data rate is not high: the 25-th, 50-th, and 75-th percentiles of the phone's throughput are 1.05Mbps, 1.98Mbps, and 5.70Mbps, respectively. Second, the watch-side throughput is even lower, with the 25-th, 50-th, and 75-th percentiles being 0.80Mbps, 1.17Mbps, and 2.31Mbps, respectively. This is likely explained by the watch's smaller radio antenna or its reduced Rx power compared to the smartphone's WiFi. Also, given that Bluetooth's peak data rate is around 2Mbps, in about 30% of the cases using BT as the pipe may throttle the wearable's subflow. This can be mitigated by the storing mode introduced in §2.3. Third, collaboratively using the two devices' WiFi interfaces leads to significant improvement of the download throughput. Figure 6 plots the distribution of the acceleration ratio, which is computed as the ratio between the wearable's throughput and the phone's throughput when they simultaneously receive data. The 25-th, 50-th, and 75-th percentiles of the acceleration ratio are 50%, 70%, and 80%, respectively. Overall, the results indicate that leveraging wearables to assist phone's network transfers is feasible and can bring significant speedup.

5. DISCUSSIONS

Implementation and Deployment. DMM can be implemented as an MPTCP extension realized in the kernel. A recent study [14] demonstrates the feasibility of realizing most of the MPTCP logic in the user space. We can also take this more flexible approach for DMM. Most of DMM's logic is transparent to the mobile apps. Some advanced features such as primary/secondary device swapping (§3) need to be controlled by applications through simple APIs. DMM should also be transparent to servers. This is indeed true if the server is MPTCP-capable. However, since not all of today's servers support MPTCP, a practical approach is to introduce an MPTCP proxy that performs translation between MPTCP (with clients) and single-path TCP (with servers). The proxy works entirely at the transport layer so it does not break TLS/SSL.

Wearable Energy Consumption. DMM naturally incurs energy overhead on wearables. We argue that in many scenarios, users do have incentives to trade a reasonable amount of wearable's energy for better performance. To gain more insights, we conduct a feasibility study for the cellular onload use case shown in Figure 1(c). We consider a typical watch's battery capacity: 570 mAh as offered by LG Watch Urbane 2. All power numbers below are obtained from a recently derived smartwatch power model for LG Urbane [11]. The dozing power of the watch is 24.3mW, which implies that at the standby mode (with the watch face display on), the watch can last for 86.8 hours. We next assume that during x percent of the overall usage period, the watch is performing cellular onload using its LTE (for the WAN connectivity) and Classic Bluetooth (for the pipe). Due to the highly bursty smartphone traffic pattern [8], we consider $x \in \{5\%, 10\%, 15\%\}$. During onload, the watch spends additional energy on LTE (1165 mW[3]), Bluetooth (111 mW), and CPU (32mW, conservatively assuming 15% CPU utilization). If onload is enabled, the watch standby time is reduced to 23.5, 13.6, and 9.6 hours for x = 5%, 10%, and 15%, respectively, which we think are still acceptable for a typical smartwatch user in particular when her phone's battery is dying. When x = 100%, a fully charged watch can provide continuous onload for 1.6 hours, equivalent to increasing the phone's battery capacity by 456 mAh (about 1/4 of an iPhone 7's battery capacity). Using WiFi will make the wearable last even longer. Overall, we believe that the above estimation, despite being very preliminary, provides incentives of using DMM in many real-world scenarios.

6. RELATED WORK

Cross-device Data Sharing. There exist several systems allowing cross-device data sharing. MicroCast [9] leverages multiple smartphones within proximity of each other to stream video cooperatively. COMBINE [1] allows nearby users to collaboratively download files over WWAN. Cool-Tether [17] provides energy-efficient cellular tethering. Unlike DMM, all above systems realize data sharing at the application layer. Mobile kibbutz [12] leverages tethering and MPTCP to allow a group

[3]The power model in [11] does not include LTE, so we use the LTE power model for Galaxy S4 LTE described in [4], assuming the signal strength of -95dBM and an amortized tail power of 5%.

of users to share cellular connectivity. DMM by contrast offers several additional advantages as described in §2.1. There also exist inverse multiplexing systems such as PRISM [10] and Horde [16] that construct a virtual high-bandwidth channel from several low-bandwidth channels. While their high-level concept is similar to DMM and to MPTCP, DMM focuses on optimizing the nodes' local communication over the pipe. Also these inverse multiplexing systems are usually heavy-weight, and require significant changes to the protocol stack such as TCP. Finally, none of the above systems has been applied to wearables, which DMM is designed for.

Mobile Multipath. Researchers have measured MPTCP performance on smartphones [2, 3], the interplay between multipath and applications such as video streaming [7] and web browsing [6], and the energy aspect [13]. In these studies, all subflows are established from the same mobile device. DMM instead considers distributing the subflows over multiple devices to facilitate their collaboration.

Cloud-assisted Sensitive Data Protection. There exist systems such as TinMan [19], CleanOS [18], and Paranoid Android [15] that leverage the cloud to protect mobile users' privacy. The high-level concept of TinMan [19] is similar to our wearable-assisted SUD protection scheme. There are however several key differences. First, TinMan stores the sensitive data in the cloud while ours only stores it on a user's personal wearable device to minimize SUD's exposure to the external world and to allow the user to manage her SUD completely locally. Second, TinMan uses code offloading to switch the control from the local device to the cloud, while ours employs HTTPS request rewriting, which is much more lightweight and ensures the minimal involvement of the wearable. Third, TinMan uses IP address spoofing to let the mobile client and the cloud "share" the same connection. This ad-hoc approach will fail, for example, when the spoofed IP packet is lost or dropped by a local router. We instead leverage DMM, a more systematical approach, to enable convenient sharing of a TCP connection.

7. CONCLUSION & ON-GOING WORK

We routinely carry both wearables and phones. In this paper, we explore the feasibility of leveraging wearables' network interfaces to enhance the phone-side network performance and other aspects such as security. Based on our field tests, under a wearable's assistance, DMM can improve smartphone WiFi download throughput by 70% (the median value) in public places. We are currently implementing a DMM prototype. Leveraging DMM, we plan to further build new applications such as the wearable-assisted SUD protection system.

Acknowledgements

We thank the WearSys reviewers for their feedback. This work was supported in part by NSF Award #1629347 and NSF Award #1618898.

8. REFERENCES

[1] ANANTHANARAYANAN, G., PADMANABHAN, V. N., RAVINDRANATH, L., AND THEKKATH, C. A. Combine: leveraging the power of wireless peers through collaborative downloading. In *MobiSys* (2007).

[2] CHEN, Y.-C., LIM, Y.-S., GIBBENS, R. J., NAHUM, E. M., KHALILI, R., AND TOWSLEY, D. A Measurement-based Study of MultiPath TCP Performance over Wireless Networks. In *IMC* (2013).

[3] DENG, S., NETRAVALI, R., SIVARAMAN, A., AND BALAKRISHNAN, H. WiFi, LTE, or Both? Measuring Multi-homed Wireless Internet Performance. In *IMC* (2014).

[4] DING, N., WAGNER, D., CHEN, X., PATHAK, A., HU, Y. C., AND RICE, A. Characterizing and modeling the impact of wireless signal strength on smartphone battery drain. In *SIGMETRICS* (2013).

[5] FORD, A., RAICIU, C., HANDLEY, M., AND BONAVENTURE, O. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824, 2013.

[6] HAN, B., QIAN, F., HAO, S., AND JI, L. An Anatomy of Mobile Web Performance over Multipath TCP. In *CoNEXT* (2015).

[7] HAN, B., QIAN, F., JI, L., AND GOPALAKRISHNAN, V. Mp-dash: Adaptive video streaming over preference-aware multipath. In *CoNEXT* (2016).

[8] HUANG, J., QIAN, F., GUO, Y., ZHOU, Y., XU, Q., MAO, Z. M., SEN, S., AND SPATSCHECK, O. An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance. In *SIGCOMM* (2013).

[9] KELLER, L., LE, A., CICI, B., SEFEROGLU, H., FRAGOULI, C., AND MARKOPOULOU, A. Microcast: Cooperative video streaming on smartphones. In *MobiSys* (2012).

[10] KIM, K.-H., AND SHIN, K. G. Improving tcp performance over wireless networks with collaborative multi-homed mobile hosts. In *MobiSys* (2005).

[11] LIU, X., CHEN, T., QIAN, F., GUO, Z., LIN, F. X., WANG, X., AND CHEN, K. Characterizing Smartwatch Usage in the Wild. In *MobiSys* (2017).

[12] NICUTAR, C., NICULESCU, D., AND RAICIU, C. Using cooperation for low power low latency cellular connectivity. In *CoNEXT* (2014).

[13] NIKA, A., ZHU, Y., DING, N., JINDAL, A., HU, Y. C., ZHOU, X., ZHAO, B. Y., AND ZHENG, H. Energy and Performance of Smartphone Radio Bundling in Outdoor Environments. In *WWW* (2015).

[14] NIKRAVESH, A., GUO, Y., QIAN, F., MAO, Z. M., AND SEN, S. An in-depth understanding of multipath tcp on mobile devices: measurement and system design. In *MobiCom* (2016).

[15] PORTOKALIDIS, G., HOMBURG, P., ANAGNOSTAKIS, K., AND BOS, H. Paranoid android: versatile protection for smartphones. In *ACSAC* (2010).

[16] QURESHI, A., AND GUTTAG, J. Horde: separating network striping policy from mechanism. In *MobiSys* (2005).

[17] SHARMA, A., NAVDA, V., RAMJEE, R., PADMANABHAN, V. N., AND BELDING, E. M. Cool-tether: energy efficient on-the-fly wifi hot-spots using mobile phones. In *CoNEXT* (2009).

[18] TANG, Y., AMES, P., BHAMIDIPATI, S., BIJLANI, A., GEAMBASU, R., AND SARDA, N. Cleanos: Limiting mobile data exposure with idle eviction. In *OSDI* (2012).

[19] XIA, Y., LIU, Y., TAN, C., MA, M., GUAN, H., ZANG, B., AND CHEN, H. Tinman: Eliminating confidential mobile data exposure with security oriented offloading. In *EuroSys* (2015).