Scheduling Virtual WiFi Interfaces for High Bandwidth Video Upstreaming Using Multipath TCP

Shobhi Maheshwari
University of Utah
shobhim@cs.utah.edu

Philip Lundrigan
University of Utah
philipbl@cs.utah.edu

Sneha Kumar Kasera
University of Utah
kasera@cs.utah.edu

ABSTRACT

Live video *upstreaming* refers to the flow of live data in the upstream direction from mobile devices to other entities across the Internet and has found use in many modern applications such as remote driving, the recent social media trend of live video broadcasting along with the traditional applications of video calling/conferencing. Combined with the high definition video capturing capabilities of modern mobile devices, live video upstreaming is creating more upstream data traffic then what present day cellular networks are equipped to support, often resulting in sub-optimal video experience, especially in remote or crowded areas with low cellular connectivity and no WiFi.

We propose that instead of using its single cellular connection, a mobile device connects to multiple nearby mobile devices and splits the live video data over the cellular bandwidth of these devices using Multipath TCP protocol. The use of MPTCP, for upstreaming live video data, has largely remained unexplored especially for scenarios where WiFi connectivity is not available. We use wireless interface virtualization, offered by Linux, to enable Multipath TCP to scale and connect to a large number of cellular devices. We design and build a system that is able to assess the instantaneous bandwidth of all the connected cellular devices/hotspots and uses the set of the most capable cellular devices for splitting and forwarding the live video data. We test our system in various settings and our experiments show that our system greatly increases the bandwidth and reliability of TCP connections in most cases and in cases where there is a significant difference in the throughput across cellular hotspots, our solution is able to recognize and isolate the better performing cellular hotspots to provide a stable throughput.

KEYWORDS

MPTCP, Virtual Wireless Interfaces, Access Points, Cellular Devices/Hotspots

1 INTRODUCTION

Cellular networks have traditionally been optimized to deliver content to mobile devices as much of the data flows in the downstream direction. However, the introduction of smartphones with high-resolution cameras has increased the demand for high-quality video content not only in downstream direction but in the upstream direction as well. Recently, live video *upstreaming* has found an application in Autonomous Vehicles where a person can control a vehicle remotely to guide it through situations that the vehicle is not equipped to handle [2]. This idea often termed as "Remote Driving" uses wireless connections to transmit the video data to the remote operator and is being employed by several companies to make the transition from human-driven to autonomous vehicles [1].

Another application of live video upstreaming is the recent social media trend of broadcasting user-generated content from mobile devices. Viewers can connect and watch the video in real-time, all the while interacting with the broadcaster in the form of comments or likes. Facebook Live is currently one of the biggest live video broadcasting services in the world, along with Periscope by Twitter and Youtube Live. Yet another example of a live video upstreaming platform is Twitch used by video gaming enthusiasts who enjoy live-broadcasting their games and watching other users play, compete, teach and participate in all gamers' activities.

Even though high bandwidth WiFi links are capable of upstreaming live video data at a high quality, they fail in a more mobile/outdoor environment. Even with various deployments, by several independent parties covering central areas of a city [10], WiFi is often limited to urban areas and is not so widely available. On the other hand cellular networks, by their very nature, provide wide-range coverage and opportunity for user mobility. Cellular base stations have a wide coverage area and therefore, have the ability to make the live video upstreaming experience truly mobile. However, cellular links are not always adequate to support video transmission because of varied reasons. First, the asymmetric nature of cellular networks places more focus on downloading content, rather than uploading. Second, the data rate of a cellular connection often varies dramatically [8] over time based on spatiotemporal variations in cellular channel conditions. Furthermore, cellular blackspots/notspots, generally defined as geographic areas that experience reduced network coverage due to physical obstructions including hills, trees, and buildings [6], can affect a particular cellular connection's data rate solely because of cell tower placement. Third, every year mobile devices are equipped with increasingly advanced cameras capable of recording high-definition videos which require higher cellular data rate.

WiFi, due to its high bandwidth, is capable of upstreaming high-definition videos but cannot provide true mobility. Cellular networks provide true mobility but fall short of upstreaming high-quality videos. To improve the mobility experience and still be able to achieve the high bandwidth attained by WiFi, several solutions

ICDCN '19, January 4–7, 2019, Bangalore, India

ACM ISBN 978-1-4503-6094-4/19/01…$15.00
https://doi.org/10.1145/3288599.3288620

have been proposed that can aggregate bandwidth by *offloading* data to nearby mobile nodes [4, 14] but these solutions do not address the challenges of upstreaming live video data especially using the Multipath TCP protocol. In this work, we address the problem of bandwidth aggregation for high-quality video upstreaming using the Multipath TCP protocol. We assume that a client (a user broadcasting a live video) is in a social environment, i.e., surrounded by people that he/she is acquainted with and can request them to share their cellular bandwidths. A client could also use socially unknown nearby devices as long as appropriate incentive mechanisms can be incorporated [19].

Multipath TCP (MPTCP) [8, 12, 22–24] is a major modification to TCP that allows multiple paths, known as subflows, to be used "simultaneously" by a single transport connection. Figure 1 shows an MPTCP connection between MPTCP-enabled client and server with 3 different subflows. An MPTCP subflow is identified by a combination of client-server IP addresses and port numbers, similar to standard TCP, and associated with a network interface connected to an access point, Cellular Hotspots (CD) in our case[1]. Essentially, MPTCP is limited to the number of physical interfaces a device can support. For example, a standard laptop has two network interfaces: one WiFi and one Ethernet interface; smartphones often have a WiFi interface and a cellular interface. Most studies associated with MPTCP have explored its utility with just two interfaces. One possible solution is interface emulation and in a recent work, Lim et al. [18] use MPTCP in a simulation framework for stored video and file transfer applications. However, to the best of our knowledge, the use of MPTCP for live video upstreaming has largely remained unexplored. In our work, we overcome the restrictions posed on the Multipath TCP protocol, by the number of wireless interfaces available on a mobile device, by virtualizing the wireless interface and scheduling these virtual interfaces in a time-shared weighted round-robin fashion to transmit data that gives more weight to the interface that is able to send more data or, in other words, has higher throughput as compared to interfaces with lower throughput.

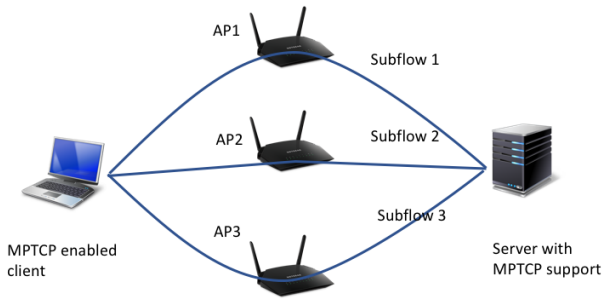

Figure 1: An MPTCP connection going through 3 different Access Points.

[1]For the rest of this paper, we will use subflow and interface interchangeably.

Another problem with the use of Multipath TCP for live video transmission is the variable throughput between the devices. The cellular bandwidth of individual neighboring devices can be different because of reasons such as distance from the base station, different carriers, etc. Although the creation of virtual interfaces provides a solution to the limited number of wireless interfaces on a device, it can not address the problem of variable throughput.

Essentially, when the difference between the throughput of two subflows is large, a large number of packets arrive out-of-order at the receiver [20]. MPTCP holds these out-of-order packets in a reordering queue at the receiving end until all the data sequence numbers are in order and therefore, a delayed packet can block all the packets with higher data sequence number that arrived before it. This problem is even more prevalent in the upstream direction because of lower upload speeds provided by the cellular networks and other shortcomings of cellular networks stated above. Therefore, in such scenarios, a better solution would be to drop the poor performing subflow. However, the complexity of the task of selecting the subflows to drop increases as the number of subflows associated with a particular transmission increase.

To complement the scalability provided by wireless interface virtualization, we design an intelligent system that adds or drops subflows based on their ability to contribute to the aggregate throughput. Our system maintains a global view (aggregate throughput) of all the associated subflows to understand how a particular subflow would perform in conjunction with other subflows and drops the subflows that could prove to be a possible bottleneck. It transmits data over every single subflow in a weighted round-robin fashion and at the end of the transmission slot for a particular subflow, the throughput achieved by it is measured and based on the comparison of the current throughput as well as the global throughput, the decision of dropping or keeping the current subflow is made.

We evaluate our system in three ways. First, we test our implementation under stable conditions in a lab with stationary cellular hotspots. Second, we set up our system in an outdoor environment and the cellular hotspots are allowed to roam free, moving in and out-of-range of the broadcasting client. Third, we deploy our system on a high-speed commuter train. Our experiments show that, in general, our system is able to aggregate the video throughput, depending on the number of cellular hotspots and their individual throughput. Our system is also able to account for delay variability between individual subflows and how that affects their respective bandwidths. Our system is able to isolate and drop low performing subflows but probes them periodically to check for any changes in the connectivity.

In summary, we make the following contributions in this paper:

- Design an intelligent system that works by dropping/adding subflows for increasing the aggregate upstream bandwidth for live video data.
- Incorporate a feedback loop that provides a server-side view of the throughput to the client, to help the client make an informed decision about which subflows to transmit on.
- Implement our globally aware dynamically adjusted algorithm in the Linux kernel. Also, implement a selection algorithm that dynamically selects an efficient subflow to initiate the MPTCP connection.

- And lastly, evaluate our design/implementation under different network conditions to test it for robustness and scalability.

2 RELATED WORK AND BACKGROUND

A promising and low-cost solution to the exponential increase in cellular data traffic download is to utilize nearby mobile devices or WiFi access points to offload cellular data [15]. MicroCast [17] is a system that uses a group of mobile users, in close proximity of each other, to cooperatively *download* a video. However, MicroCast does not deal with live video data. Also, as noted above, MicroCast works with downstream data whereas our system deals with the less researched problem of data upstreaming. Quality Aware Traffic Offloading (QATO) [15] enables a user with a poor cellular connection to offload its data, with the help of a base station, to another nearby user with a better cellular connection for helping to transfer the data to the internet. QATO suggests the use of Wi-Fi Direct for transmission of data from the source node to the neighboring node. Although QATO deals with uploading data, it only uses a single mobile node from its neighbors to offload data. Also, QATO works with stored data, such as pictures and text files, for uploading and does not consider live video data. [11, 25] also deal with cellular offloading using nearby devices but work primarily with video streaming/downloading.

Similar to our system, *mobiLivUp* [19] utilizes nearby smartphones and their cellular bandwidth to effectively increase the live video upstream bandwidth. mobiLivUp works by creating a small wireless network, using WiFi Direct that nearby devices can then connect to. The video stream is split into multiple different streams and then sent to these connected devices to be uploaded to the server through the devices' network connection. However, mobiLivUp requires an application layer splitter and gatherer for handling the multiple data streams which can limit its capability to work with unmodified video broadcasting systems. Instead, our system takes all the "splitting" and "gathering" complexity out of the application layer into the operating system and together with MPTCP, it is compatible with any application the user might prefer.

Stream Control Transmission Protocol (SCTP) [21] can be used for transmitting multiple streams of data in parallel between two end points that have an established network connection. However, SCTP does not have the capability to scale to more than one wireless interfaces and is not necessarily optimized for live video data. In contrast to SCTP, Multipath TCP [12] works with multiple streams of data and has a built-in ability to scale to a large number of wireless interfaces.

2.1 Multipath TCP

Multipath TCP, as proposed by the IETF working group *mptcp* [12], is an effort towards extending the functionality of standard TCP by spreading a single data stream across different interfaces (e.g., WiFi and LTE on a smartphone). Multipath TCP can utilize the throughput achievable over every available interface thereby increasing the aggregate throughput for the application. Multipath TCP appears as a regular stream-socket interface to the application, however below the application layer, TCP subflows are created for each interface. The multiple subflows going through the different interfaces in combination form a Multipath TCP connection and use TCP options for signaling the necessary control information between the end hosts. Since every subflow is similar to a standard TCP connection between two endpoints, Multipath TCP appears to be a regular TCP connection to the firewalls/middleboxes along the subflows' paths. Thus, Multipath TCP works with unmodified applications and is deployable in today's Internet [24].

The benefits of using multiple paths to transmit data include better resource utilization, better throughput, increased redundancy and smoother reaction to failures as the connection can still persist through other paths when a path fails. MPTCP also has benefits of load balancing for multihomed servers and data centers, and for mobility [29]. Many industry leaders such as Apple and Samsung have already adopted this protocol in their latest iOS and Android systems. The iOS operating system now uses MPTCP to optimize the delay-sensitive traffic generated by Siri, Apple's Personal Assistant [3].

2.2 Wireless Network Interface Virtualization

Wireless Network Interface Virtualization is a concept that involves establishing and maintaining concurrent Access Point (AP) connections for better robustness and throughput. *Spider* [26, 27] is a system that uses multi-AP selection, channel-based scheduling, and opportunistic scanning to maximize throughput while mitigating the overhead of association and DHCP. *WiSwitcher* [13] virtualizes the wireless driver sitting on top of the single radio card, such that it appears as independent Virtual Stations associated with their respective Access Points. The ViFi project [5] exploits the use of multiple APs to improve the link layer performance for common applications such as Web browsing and VoIP, for moving vehicles. MultiNet [7], FatVAP [16], and PERM [28] are some more examples of systems that enable clients to associate with more than one nearby Base Station (BS), to increase throughput if the wireless capacity is greater than the capacity of wired links behind the BSes.

Thus, the idea of Wireless Network Interface Virtualization and associating to multiple APs for bandwidth aggregation is not new and has been studied quite extensively but what is missing is the ability of an unmodified application to benefit from this bandwidth aggregation, as standard TCP is designed to work with a single interface by default. Multipath TCP enables unmodified applications to use these interfaces. Niculescu et al. [10] make some efforts towards making unmodified applications Multipath capable. Their system name MultiWiFi leverages Multipath TCP to achieve seamless mobility in WiFi, by letting a client connect to multiple APs on the same channel and splitting the traffic between them using Multipath TCP, thereby enabling a WiFi client to achieve close to the maximum achievable throughput in a wide range of scenarios. However, Multi-WiFi has been designed mainly for downstreaming content and is not optimized for live video data. For this reason, MultiWiFi is ill-equipped to handle the challenges faced with moving video data traffic from the client to the server. We design a system that lets applications use Multipath TCP without any modification to the application with emphasis on live video transmission which is not tolerant of delays.

3 METHODOLOGY

In this section, we present our approach to efficiently aggregate the upstream bandwidth of multiple nearby devices for live videos. We design our system incrementally with the help of two algorithms: Algorithm 1 that utilizes client-side information, including round-trip time and buffer sizes, to calculate subflow throughput, and Algorithm 2 that builds on top of Algorithm 1 and introduces a feedback loop that conveys the number of bytes received at the server-side back to the client. This helps the client account for delays caused in the network and calculate server-side throughput more accurately.

MPTCP works by establishing an initial connection over a single interface and starting data transmission over this initial subflow. It then adds subflows using the MP_JOIN option. MPTCP adds subflows one at a time and sends as many packets with MP_JOIN option as there are interfaces available on an end-device. In Figure 2, the live broadcasting device has two virtual interfaces, so the MPTCP connection would be established over one of them and then the next subflow would be added later. In our system, we use the MPTCP *fullmesh* path manager, which means that MPTCP would create subflows equal to the product of the number of interfaces present on both sides. In Figure 2, the client/live broadcasting device has 2 virtual interfaces and the video server has a single interface. So, the fullmesh path manager would create $2 * 1$ subflows.

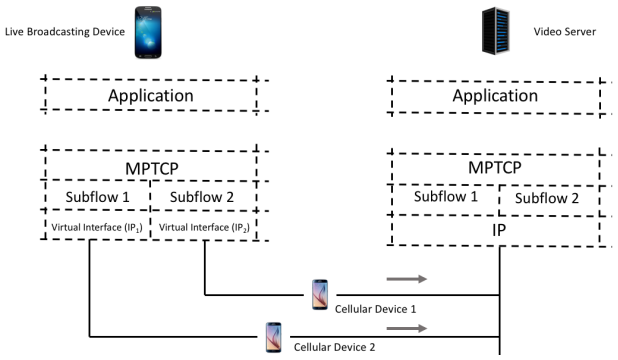

Figure 2: MPTCP connection when two virtual interfaces are present on the client-side.

A physical wireless card is capable of supporting hundreds of Mbps data rate but the actual data rate for any data transmission is determined by the rates that the network can support. More specifically, the data transmission rate is decided by the slowest/bottleneck link inside the network. So, virtualizing a physical wireless card into multiple interfaces does not necessarily affect its capability to transmit data but enables us to transmit more data by splitting the data over multiple interfaces. Therefore, Multipath TCP can be used in conjunction with virtual interfaces to establish multiple subflows between the client and the server. However, when the virtual interfaces are connected to different cellular hotspots they get different throughput. Let us consider the example of an MPTCP connection with two subflows, where $Subflow_1$ has a much higher throughput than $Subflow_2$, i.e., packets sent over $Subflow_1$ would arrive earlier than packets sent over $Subflow_2$. MPTCP scheduler is designed such that it would push more data towards $Subflow_1$ and only a small amount of data would be sent over $Subflow_2$. Since, $Subflow_2$ has poor throughput, the packets sent over $Subflow_2$ would take much longer than packets sent over $Subflow_1$. MPTCP, like TCP, ensures that packets are passed to the upper application layer in the correct order. So, while the packets sent over $Subflow_2$ are in-flight, i.e., are traveling in the network, the packets sent over $Subflow_1$ with higher sequence numbers would be held in the reordering buffer. Also, while the server is waiting for the packets sent over $Subflow_2$ it would continue to ask the client for those packets, causing the client to move to the fast retransmission state. The high mismatch in the throughput across two subflows thus results in an aggregate throughout that can be much lower than what we could achieve by using only $Subflow_1$. Therefore, in our system, we aim at increasing the aggregate bandwidth by dropping subflows that can result in an increase in the number of the *out-of-order* packets.

Furthermore, in the network, every MPTCP subflow is treated as an individual TCP flow. Therefore, to maintain the backward compatibility of MPTCP with TCP (if one of the end hosts does not support MPTCP, then the connection should fall back to regular TCP) every subflow has an associated sending/receiving buffer and the end hosts maintain data structures containing metadata about the subflow, similar to the metadata maintained for a standard TCP connection. We leverage this information to obtain a server-side view of all the MPTCP subflows for more accurate throughput calculations at the client-side.

In this section, we first propose a base algorithm that uses the round trip time, *rtt*, and *buffer length* maintained at the client-side (i.e., the user broadcasting live video) to estimate the throughput at the server-side (video server in the cloud where the live video is being uploaded). However, the throughput estimated at the client can differ completely from what the receiver sees because of the delays introduced inside the network and out of order packets. Therefore, in our final algorithm, we introduce a feedback loop to bridge the gap between the client's and the server's views.

3.1 Algorithm 1: Client-Side Throughput Calculation

An MPTCP connection maintains information about the packet round-trip times and the transmission buffer length at the sender-side (client-side, in our case) for every subflow. Therefore, the throughput calculations are often done on the sender-side. We use these calculations in our first approach.

The selection of subflows can be done in two ways. First, we can measure the throughput for the interfaces in advance, before starting the transmission and then add subflows based on the throughput using MP_JOIN. Second, we can start the transmission with subflows equal to the number of active interfaces we have and then drop the subflows that prove to be a bottleneck later. The first approach works well in the scenarios where we have only two interfaces, as in that case, we can just select one interface if the difference between the throughput is large. However, this approach does not scale well and the complexity of selecting the interfaces to start the connection with increases as the number of active interfaces increase. Also, in our scenario, the cellular hotspots are

mobile in nature and are at liberty to move in and out-of-range of the broadcasting client. In such a dynamic environment, selecting a set of 'good' interfaces for transmission can prove to be a challenging task. A wrong selection can result in less than optimal throughput. Lastly, because of the mobile nature of the cellular hotspots, a poor performing subflow can recover later with time. Therefore, we decide to make a bottleneck subflow a *backup* flow instead of completely dropping it, as explained in Section 3.2.

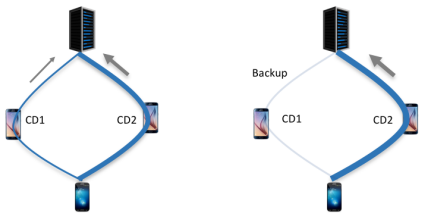

Figure 3: Scenario 1 (left) : The client connected to CD_1 and CD_2 would push more data towards CD_2 because of its better channel condition. Scenario 2 (right) : CD_1 moves away from the client and the client moves all the traffic to CD_2 while keeping CD_1 as a backup.

We begin with the assumption that every subflow is a good flow, i.e., it can increase the aggregate throughput of the entire connection. By making this assumption, we are giving each subflow a fair chance for consideration. Basically, each subflow is initially scheduled to transmit some data, which in turn helps to better evaluate its performance. Consider the two scenarios in Figure 3. Even though CD_2 is clearly the better choice than CD_1 because of its better channel conditions, the client still chooses to transmit some data over CD_1 but as CD_1 moves away from the client, the client makes the decision of dropping the CD_1 subflow. For every subflow, we measure the throughput of the subflow based on the buffer length or maximum segment size (*mss*) and the round trip time, *rtt*. At the same time, we also keep track of the global throughput, i.e., the throughput achieved over all the subflows. If the throughput for the subflow threshold is less than a certain fraction of the global average we drop that subflow so that it is not be scheduled the next time its turn comes up.

Initially, the client mobile device starts transmitting the live stream using MPTCP over the first subflow and MPTCP adds subflows without any interference from our system. The system schedules interfaces from a queue, i.e., brings them *on-channel* and a particular interface stays on-channel as long as the subflow buffer has data to transmit. Once the buffer is emptied, the interface is taken *off-channel* and a null frame is sent to the CD, to inform the CD that the interface has entered power save mode (PSM) and to buffer any packets for the client. When this same interface comes back on-channel again, it will inform the CD and the CD will deliver all the buffered packets to the interface.

Before an interface goes off-channel, we sample its throughput and compare it to the global throughput of the MPTCP connection. If the throughput associated with a particular CD_i is less than a certain fraction of the global throughput,

$$B_{CD_i} < \alpha B_{gbl} \tag{1}$$

where α is the throughput threshold, then the interface is marked inactive, moved to the *probe_list* and the *probe_timer* is set. When the probe_timer expires all the interfaces from the *probe_list* are moved to the *active_list* and their performance is evaluated once again.

Algorithm 1 BaseApproach

```
1:  srtt ← 0                                          ▷ Smoothed RTT calculated over time
2:
3:  procedure CalculateThroughput(rtt, mss)           ▷ Instantaneous RTT
4:      B ← 0
5:      β ← 0.125
6:      if srtt! = 0 then
7:          srtt = (1 − β) ∗ srtt + β ∗ (rtt)
8:      else
9:          srtt = rtt ∗ 8
10:     B = mss/srtt
11:     return B
12:
13: procedure CDSwitch( )
14:     B_gbl ← 0
15:     β ← 0.125
16:     for CD_i in CD_list do
17:         B_i ← CalculateThroughput(rtt, mss)
18:         if B_i < α ∗ B_gbl then
19:             // Mark CD_i as backup
20:             if probe_timer is not set then
21:                 // set probe_timer
22:         if probe_timer == 0 then                  ▷ probe_timer expired
23:             for CD_i in CD_list do
24:                 if CD_i is backup then
25:                     // Mark CD_i as active
26:         B_gbl = (1 − β) ∗ B_gbl + β ∗ B_i          ▷ EWMA calculation
27:     return
```

In Section 4.1, we evaluate this algorithm under different scenarios to determine the optimal value of α, the throughput threshold used in the procedure CDSwitch. The use of exponentially weighted moving average (EWMA) for the calculation of the global throughput ensures that as the algorithm circles through the list of interfaces, the influence of old samples falls exponentially as new samples are added to the average. Another thing to note here is that even when a particular interface performs poorly during its turn, we still add its throughput to the global average. This is quite useful in scenarios where, for instance, if the client moves away from the set of CDs it is connected to. In this case, the global average throughput would go down as the next interfaces are scheduled but the algorithm would still be able to adapt to the worse but new network conditions.

3.2 Rationale Behind *probe_lists*

Since every device involved is mobile (the client requesting the connection and the device acting as the access point are all cellular devices), it may happen that a particular CD moves out-of-range, i.e., away from the client device (such that its throughput is less than the average throughput) but it may also move back in-range sometime in the future. So, while trying to increase our aggregate

throughput, if we disconnect when a device moves out-of-range and try to reconnect when it comes back in range, we might end up wasting a significant amount of time (up to 15 seconds [9] because of the DHCP and the WEP/WPA key exchanges involved) in connecting to a CD. Since re-associating with a particular CD can have such high-performance penalties, in practice, we move the interface associated with that particular CD to the "*probe_lists*" and check for any changes in the bandwidth of the members of the list periodically. If the bandwidth increases then we can avoid all the hassle of a new association and save the time we would have otherwise wasted in trying to reconnect to the CD.

Also, in a truly mobile scenario, there is no accurate way of predicting the throughput the client will obtain from a CD in the very near future and establishing a new connection every time is a decision that locks the device to one CD for a certain period of time resulting in sub-optimal performance. Therefore, it makes sense to stick with a choice for some amount of time, in case the CD may be facing some temporary short-term fluctuations that are affecting its capacity.

3.3 Algorithm 2: Globally Aware Dynamically Adjusted

The Internet is a best-effort network, meaning that the packets will be delivered if possible, but may also be dropped. Therefore, our first approach with client-side throughput calculations, may work well in theory but in practice, these dropped packets have to be retransmitted by MPTCP and this retransmission reduces the throughput on the server-side for two reasons:

- The lost data needs to be sent again, which consumes time. The delay introduced by this retransmission is inversely proportional to the rate of the bottleneck link, i.e., the slowest link in the network between the sender and the receiver.
- The MPTCP protocol uses *acknowledgments* as a means of feedback about what packets were delivered. Detection of undelivered packets relies heavily on these acknowledgments. Due to propagation delays, acknowledgments can only be received by the sender with some latency, which further impacts transmission. In most practical scenarios, this is the most significant contribution to the extra delay caused by the retransmission.

So, for the precise calculation of the throughput, these factors need to be taken into consideration. The MPTCP rate is regulated by its congestion window size, slow-start duration, sender window (and receiver window) size. A suboptimal configuration of these variables will make the subflow throughput measured a bit lower. More importantly, especially in wireless channels, the channel condition changes also lead to adaptation of MPTCP congestion and flow controls. Such adaptation can also lead to the throughput being calculated differently than the actual value. All these conditions would lead to the sender and the receiver seeing two completely different sets of throughput and delay. The throughput calculation at the sender-side, while giving a pretty good estimate of the receiver-side throughput, does not reflect the exact throughput that the receiver perceives.

The receiver has a more accurate global understanding of all the subflows whereas the sender is responsible for managing the interfaces. To bridge this gap between the sender and the receiver we introduce a feedback loop from the receiver to the sender. The feedback can now be used in two ways to improve our base algorithm, which we describe next.

3.3.1 ***Approach 1: Globally Aware Subflow Selection.*** As discussed above, the things that affect the throughput of a particular subflow are *mss*, *rtt*, congestion window, retransmission rate (loss rate) and other network delays. The client-side calculates the *rtt* but the server has a better understanding of how many bytes are received on which subflow. So, the server sends this information to the client as feedback. This information is embedded in the TCP acknowledgment headers' options field which can then be read by the client when the acknowledgment is received. MPTCP packets are similar to TCP packets in the sense that an MPTCP packet would have the same TCP header as a regular TCP packet but with the MPTCP control information in the TCP options field.

The objective of this algorithm is also to optimize the value of α, the throughput threshold, in the procedure CDSwitch, in algorithm 1.

3.3.2 ***Approach 2: Subflow Selection based on Adaptive Throughput Threshold.*** The disadvantage of using a fixed threshold is that it will treat every connection similarly, whereas it can happen that a certain threshold while being perfectly reasonable for a particular transmission may be too high for another transmission and can lead to loss of throughput or it may be too low for yet another transmission and slow down the whole connection. So, the challenge of which subflow is a poor-performing subflow and how many such subflows exists remains.

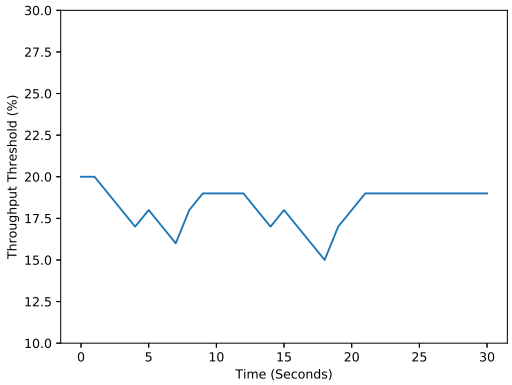

Figure 4: The throughput threshold decreases step by step every time a AP contributes to the global throughput but it increases based on the mean of the α_{curr} and α_{max}

To mitigate this problem, we follow an approach to dynamically adjusted α. We start with a high value of α and decrease the value of α every time B_{gbl} increases or remains the same. However, when we see a decrease in the B_{gbl}, we set the value of α to the mid-point

of the current value of α and the initial value of α. Algorithm 2 gives a pseudo-implementation of our dynamically adjusted approach. Setting the value of α as the mean of its current value and its max value ensures that α never exceeds α_{max}. Figure 4 illustrates the variation of α corresponding to one of the experiments explained in section 4.3.

Algorithm 2 Globally Aware Dynamically Adjusted Approach

```
1:  procedure CDSwitch( )
2:      B_gbl ← 0
3:      β ← 0.125
4:      α_max ← 0.20
5:      α ← α_max
6:
7:      for CD_i in CD_list do
8:          B_i ← CalculateThroughput(rtt, bytes_received)
9:          B_iexp ← CalculateThroughput(rtt, mss)
10:         if B_iexp < B_i & B_i >= α * B_gbl then
11:             α − = 1
12:         else if B_iexp > B_i & B_i >= α * B_gbl then
13:             //Do Nothing
14:         else
15:             α = (α + α_max)/2
16:             // Mark CD_i as backup
17:             if probe_timer is not set then
18:                 // set probe_timer
19:
20:         B_gbl = (1 − β) * B_gbl + β * B_i                ▷ EWMA calculation
21:         // Switch to the next active interface
22:
23:         if probe_timer == 0 then                          ▷ probe_timer expired
24:             for CD_i in CD_list do
25:                 if CD_i is backup then
26:                     // Mark CD_i as active
27:     return
```

3.4 Initial Sub-flow Selection

When multiple interfaces are available, the Linux operating system selects a particular interface as the primary interface. The problem with a static primary interface arises when the primary interface is connected to a bad CD or a CD that is too far away. If the path through the primary interface is congested or the rtt through it is quite high then MPTCP makes a few attempts to establish the connection but ultimately gives up and returns a failure.

In this case, the MPTCP connection can never be established as the first MPTCP subflow is created over the primary interface. To solve this problem, we have implemented a simple algorithm that can dynamically scan all the interfaces and establishes the MPTCP connection over the first working interface. To find a working interface, the algorithm cycles through the list of interfaces starting with the primary interface and sends a few *echo request* messages to the receiver over every interface. The first interface it gets an *echo response* on would be used for connection establishment. While scanning through the list of interfaces, the algorithm looks for the first working interface rather than the best interface because MPTCP would ultimately create subflows through all the interfaces and we don't need to waste our efforts trying to find the best interface.

4 EVALUATION

We have implemented our globally aware dynamically adjusted algorithm in the Linux 3.14.0 kernel, patched with MPTCP v0.89. Most of the changes are made to the 802.11 module of the Linux kernel with Atheros 9K NIC card as the wireless card, although the changes are not dependent on any NIC card.

In this section, we evaluate each of our contributions experimentally, in an indoor as well as an outdoor environment to account for different network conditions and test our system for scalability and robustness. First, we evaluate our system with different throughputs and delays to find an optimal value for the throughput threshold. Next, we run tests with throughputs above and below the throughput threshold, to analyze how our system handles the poor performing subflow. Then, we run experiments to test our system in a comparatively stable indoor and outdoor environments. Finally, we run our system with two different cellular links in a real-world live video transmission in a commuter train to test its performance in a real-world scenario.

4.1 Calculating Throughput Threshold

In this section, we run tests with different throughputs and delays, to figure out an optimal value of the throughput threshold, so that our system is able to differentiate between a subflow that would contribute to the aggregate throughput compared to a bottleneck subflow. To create unbalanced network conditions, we use `linux traffic control (tc)` utility to control the egress throughput (upload throughput) over a single subflow while letting the other subflow be uncontrolled. During our experiments, we are connected to the on-campus WiFi on both the subflow, which can support upload speeds up to 14 Mbps. We also performed these experiments while connected to two different cellular devices (CDs), to account for the real use case of our problem statement.

Our implementation compares the throughput of a particular interface with the global average just before switching to the next CD. Switching from one CD to the other requires transmitting a null IEEE 802.11 frame to the CD with the power-save mode (PSM) bit set, indicating that the client is entering PSM mode. This tells the CD to buffer any packets destined for the client. Then the algorithm updates the SSID and MAC address of the other CD along with encryption parameters if they have changed. If the CDs are on different channels, the device driver is also set to the channel frequency of the next CD. After this, the device is ready to transmit through the second CD.

We evaluated our algorithm by creating two virtual interfaces and connecting to two different access point through them. For interface 1, we were connected to our campus WiFi and were able to receive a pretty stable upload throughput of almost 14 Mbps but for interface 2, we connected to a cellular device and limited the upload bandwidth to 75%, 50%, 30%, 20%, 15% and 10% of the maximum throughput on the interface 1. By emulating such unbalanced rate conditions, we want to emulate situations where one CD is providing consistently good performance whereas the performance of the other CD goes down with time. Our analysis shows that the number of out-of-order packets increases as the difference between the throughputs of the two subflows increase. For the purpose of this analysis, we used data files with size ranging from 100 KB to 100 MB. Table 1 summarizes some of the results for this experiment.

MPTCP comes with a smart built-in scheduler that pushes data to the subflow with the shortest *rtt* until its send buffer is full but

Subflow 1 (Mbps)	Subflow 2 (Mbps)	Aggregate Throughput (Mbps)
13.86	1.94	12.05
12.1	1.63	10.63
14.52	2.08	13.23

Table 1: When one subflow is less than 20% capacity of the other subflow, aggregate throughput decreases.

Subflow 1 (Mbps)	Subflow 2 (Mbps)	Aggregate Throughput (Mbps)
4.13	2.05	6.05
3.98	2.57	6.39
4.23	1.97	6.19

Table 2: When the difference between between the throughput of two subflows is not that large, throughput aggregation is achieved.

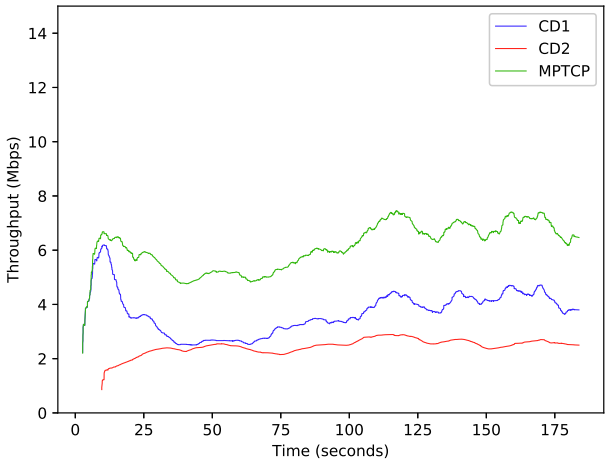

Figure 5: MPTCP combines throughput when both subflows are performing well.

MPTCP does not take into account the effect of out-of-order packets when the difference between the throughput of the subflows is large and therefore would still end up sending some data over the bottleneck subflow. The delay introduced by out-of-order packets may not be significant when the video duration is small, say a couple of seconds. But with the increase in video duration, this delay affects the overall throughput achieved at the receiver end.

Next, we run the same experiments while being connected to two different CDs through our virtual interfaces. In this case, the highest upload throughput that we were able to achieve on a single interface was around 5 Mbps and we limited the upload bandwidth of the interface 2 to 75%, 50%, 30%, 20%, 15% and 10% of interface 1. We used the same data files for this experiment as well. A summary of these experiments can be found in Table 2.

From the wide range of experiments that we performed, we found that a throughput threshold of 0.2 (20%) works best to differentiate good interfaces from the bad ones.

4.2 Base Algorithm vs Base Implementation

After finding the optimal value for the throughput threshold, we plugged this value of α in algorithm 1 and evaluated system performance. To demonstrate the function of our base algorithm, we ran experiments under following conditions: we created two virtual interfaces with one connected to on-campus WiFi access point with upload throughput around 14 Mbps and connected the second interface to a CD that had a variable upload throughput ranging from 4 Mbps to 1.6 Mbps. We ran the experiments for extended periods of time to account for a large amount of data being uploaded during a live video transmission that may last up to tens of minutes or more. Figure 5 shows that our system is able to combine the throughput of the two subflows and able to achieve more than 95% of the sum of the individual flows.

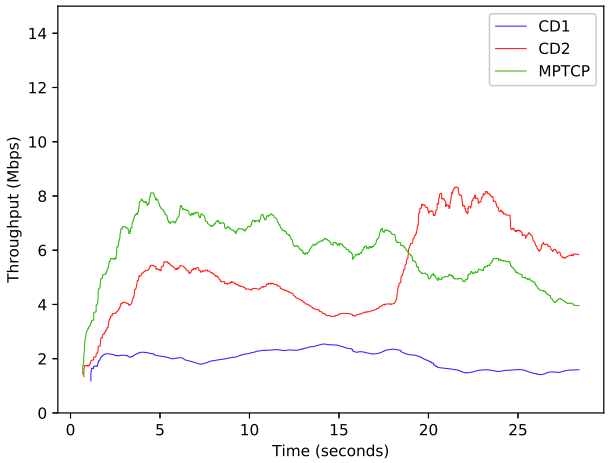

Figure 6: MPTCP througput goes down when one of the subflows is not performing well.

Next, we test the baseline system under unbalanced network conditions where on one interface we are connected to a high-speed WiFi but for the other, we are connected to a congested cellular network. Figure 6 shows how the MPTCP performance went down as the performance of the better interface went up. We next ran our system under similar conditions and, as shown in Figure 7, our system successfully recognized the bad interface and stop transmitting on it. Since, we periodically probe the backup interfaces for improvements, in Figure 7, the second interface again comes online at about 17 secs but is again shut down for its low performance.

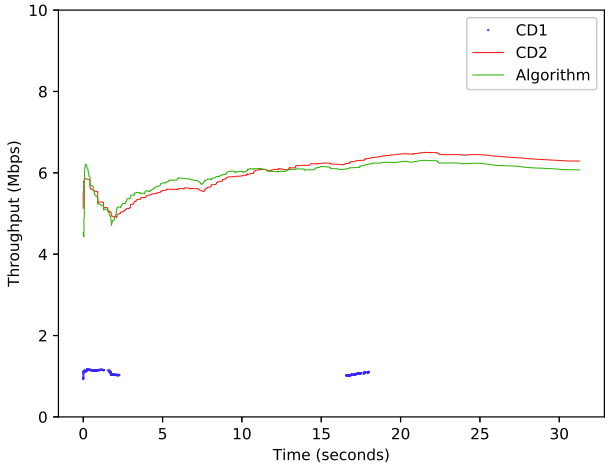

Figure 7: The system pushes all the data towards CD_2 and puts CD_1 on backup.

We perform the same experiments while being connected to two different cellular hotspots and obtained similar results as shown in Figure 6. Our base algorithm was successfully able to recognize and isolate low performing subflows to maintain the throughput of the better performing subflows.

4.3 Feedback loop vs Base algorithm

In this section, we test our implementation that introduces the feedback loop into our algorithm to see how it performs under different network conditions. Along with incorporating the feedback in our throughput calculations, we adjust our throughput threshold based on the feedback we receive from the server. We decrement the throughput threshold every time the received feedback indicates that the server-side throughput has increased or has at least remained the same but if the feedback indicates that the server-side throughput has decreased because of a particular subflow, we block that interface and increment our throughput threshold to account for the feedback.

We first test our systems in an indoor setting with two virtual interfaces connected to two cellular CDs. The testing conditions in the indoor environment were stable but we still saw an improvement over our base algorithm. We then perform the experiments in an outdoor environment, with stationary as well as mobile cellular CDs. These experiments performed similar to the indoor environment and were also able to select subflows in order to maximize the throughput achieved. As can be seen in Figure 8, initially when only CD_1 is present, the MPTCP connection closely tracks the throughput of CD_1 and when CD_2 comes online, the system is able to combine the throughput of both.

Lastly, we went on to a commuter train in our city and evaluated our systems in unstable network conditions as experienced in a fast-moving vehicle. Performance in a fast-moving vehicle is a challenge because a cell phone moves through the coverage range of a lot of

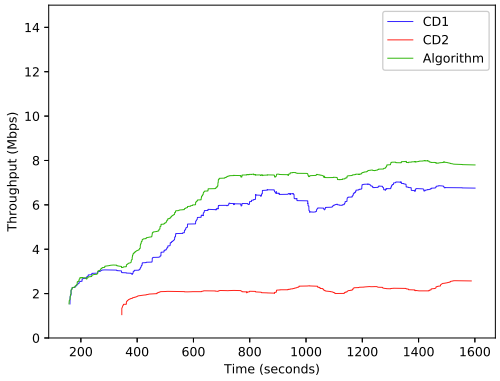

Figure 8: Shows the throughput obtained, based on the cellular traces, for individual CDs as well as the combined system throughput.

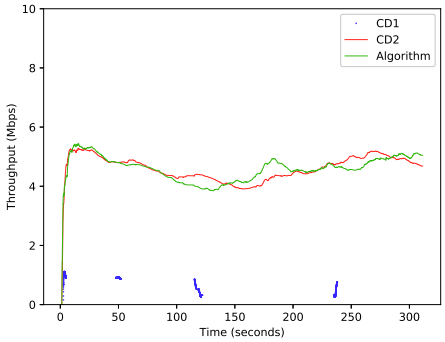

Figure 9: Throughput as calculated from the cellular traces collected on a commuter train.

base station experiencing frequent handoffs which can affect its performance and the data rates that it can achieve.

We traveled on the train while going north, outside the city, on the commuter train and the further north we went, the data rates of the cellular CDs became more and more unpredictable but our system was still able to closely track the best throughput that the two interfaces could provide. Figure 9 shows the cellular traces for the tests we ran on the commuter train. Both the interfaces are suffering from bad connectivity but one is worse than the other. Because of bad network conditions, one the working subflow often goes into slow start mode, as can be seen at around 100 secs, but the system is still able to trace the throughput of the better performing interface. The bad network conditions also increased the number of retransmission required by the protocol, as shown in Figure 10.

5 CONCLUSION & FUTURE WORK

In this paper, we address the challenge of increasing the upstreaming throughput by aggregating the throughput of nearby cellular

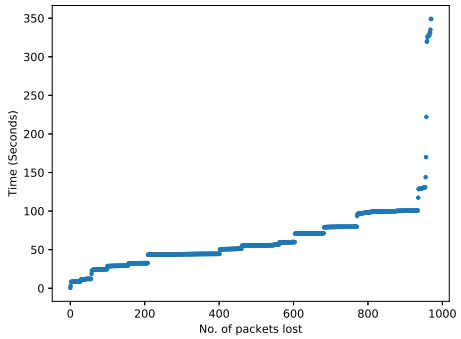

Figure 10: Shows the number of retransmissions based on the cellular traces obtain from the train experiments.

devices. We propose a novel algorithm that schedules the virtual wireless interfaces associated with a particular MPTCP connection to maximize the throughput perceived at the server-side. Our system works based on the feedback it receives from the server and uses that feedback to add or drop subflows by means of a dynamically adjusted threshold. We have implemented our algorithm in the Linux kernel and evaluated it under different network conditions. Through our evaluations, we show that we are able to achieve close to optimal throughput for situations where the throughput of the subflows was close to each other. For situations where a huge difference exists in the throughput of the subflows, our algorithm was successfully able to recognize and isolate the low performing subflow to maintain a higher throughput with the help of better performing subflows.

Our research can progress along the following directions: we plan to examine network coding methods to deal with the out-of-order packets although these methods are not likely to entirely eliminate the need for dropping poorly performing subflows. We also need to evaluate our system when multiple WiFi channels are used.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1302688.

REFERENCES

[1] [n. d.]. Huawei Demonstrates 5G-based Remote Driving with China Mobile and SAIC Motor. http://www.huawei.com/en/press-events/news/2017/6/5G-based-Remote-Driving.

[2] [n. d.]. I rode in a car in Las Vegas that was controlled by a guy in Silicon Valley. https://www.technologyreview.com/s/609937/i-rode-in-a-car-in-las-vegas-its-driver-was-in-silicon-valley/.

[3] [n. d.]. Use Multipath TCP to create backup connections for iOS. https://support.apple.com/en-us/HT201373.

[4] Adnan Aijaz, Hamid Aghvami, and Mojdeh Amani. 2013. A survey on mobile data offloading: technical and business perspectives. *IEEE Wireless Communications* 20, 2 (2013), 104–112.

[5] Aruna Balasubramanian, Ratul Mahajan, Arun Venkataramani, Brian Neil Levine, and John Zahorjan. 2008. Interactive wifi connectivity for moving vehicles. *ACM SIGCOMM Computer Communication Review* 38, 4 (2008), 427–438.

[6] Mridul Mohan Bharadwaj and Jyotirmoy Karjee. 2016. Improved Cell Coverage in Hilly Areas using Cellular Antennas. *International Journal of Advanced Networking and Applications* 7, 6 (2016), 2953.

[7] Ranveer Chandra and Paramvir Bahl. 2004. MultiNet: Connecting to multiple IEEE 802.11 networks using a single wireless card. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, Vol. 2. IEEE, 882–893.

[8] Yung-Chih Chen, Yeon-sup Lim, Richard J Gibbens, Erich M Nahum, Ramin Khalili, and Don Towsley. 2013. A measurement-based study of multipath tcp performance over wireless networks. In *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 455–468.

[9] Yung-Chih Chen, Erich M Nahum, Richard J Gibbens, Don Towsley, and Yeon-sup Lim. 2012. Characterizing 4g and 3g networks: Supporting mobility with multi-path tcp. *University of Massachusetts Amherst, Tech. Rep* (2012).

[10] Andrei Croitoru, Dragos Niculescu, and Costin Raiciu. 2015. Towards Wifi Mobility without Fast Handover.. In *NSDI*. 219–234.

[11] Savio Dimatteo, Pan Hui, Bo Han, and Victor OK Li. 2011. Cellular traffic offloading through WiFi networks. In *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*. IEEE, 192–201.

[12] Alan Ford, Costin Raiciu, Mark Handley, and Olivier Bonaventure. 2013. *TCP extensions for multipath operation with multiple addresses*. Technical Report. IETF.

[13] Domenico Giustiniano, Eduard Goma, Alberto Lopez, and Pablo Rodriguez. 2009. WiSwitcher: an efficient client for managing multiple APs. In *Proceedings of the 2nd ACM SIGCOMM workshop on Programmable routers for extensible services of tomorrow*. ACM, 43–48.

[14] Bo Han, Pan Hui, VS Anil Kumar, Madhav V Marathe, Jianhua Shao, and Aravind Srinivasan. 2012. Mobile data offloading through opportunistic communications and social participation. *IEEE Transactions on Mobile Computing* 11, 5 (2012), 821–834.

[15] Wenjie Hu and Guohong Cao. 2017. Quality-aware traffic offloading in wireless networks. *IEEE Transactions on Mobile Computing* 16, 11 (2017), 3182–3195.

[16] Srikanth Kandula, Kate Ching-Ju Lin, Tural Badirkhanli, and Dina Katabi. 2008. FatVAP: Aggregating AP Backhaul Capacity to Maximize Throughput.. In *NSDI*, Vol. 8. 89–104.

[17] Anh Le, Lorenzo Keller, Hulya Seferoglu, Blerim Cici, Christina Fragouli, and Athina Markopoulou. 2014. MicroCast: Cooperative video streaming using cellular and D2D connections. *arXiv preprint arXiv:1405.3622* (2014).

[18] Yeon-sup Lim, Yung-Chih Chen, Erich M Nahum, Don Towsley, and Richard J Gibbens. 2014. Improving energy efficiency of mptcp for mobile devices. *arXiv preprint arXiv:1406.4463* (2014).

[19] Philip Lundrigan, Mojgan Khaledi, Makito Kano, Naveen Dasa Subramanyam, and Sneha Kasera. 2016. Mobile Live Video Upstreaming. In *Teletraffic Congress (ITC 28), 2016 28th International*, Vol. 1. IEEE, 121–129.

[20] Hyunwoo Nam, Doru Calin, and Henning Schulzrinne. 2016. Towards dynamic mptcp path control using sdn. In *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*. IEEE, 286–294.

[21] Lyndon Ong. 2002. An introduction to the stream control transmission protocol (SCTP). (2002).

[22] Christoph Paasch, Gregory Detal, Fabien Duchene, Costin Raiciu, and Olivier Bonaventure. 2012. Exploring mobile/WiFi handover with multipath TCP. In *Proceedings of the 2012 ACM SIGCOMM workshop on Cellular networks: operations, challenges, and future design*. ACM, 31–36.

[23] Christoph Paasch, Ramin Khalili, and Olivier Bonaventure. 2013. On the benefits of applying experimental design to improve multipath TCP. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 393–398.

[24] 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 *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 29–29.

[25] Filippo Rebecchi, Marcelo Dias De Amorim, Vania Conan, Andrea Passarella, Raffaele Bruno, and Marco Conti. 2015. Data offloading techniques in cellular networks: A survey. *IEEE Communications Surveys & Tutorials* 17, 2 (2015), 580–603.

[26] Hamed Soroush, Peter Gilbert, Nilanjan Banerjee, Mark D. Corner, Brian Neil Levine, and Landon P. Cox. 2011. Spider: improving mobile networking with concurrent wi-fi connections. In *SIGCOMM*.

[27] Hamed Soroush, Peter Gilbert, Nilanjan Banerjee, Brian Neil Levine, Mark D. Corner, and Landon P. Cox. 2011. Concurrent Wi-Fi for mobile users: analysis and measurements. In *CoNEXT*.

[28] Nathanael Thompson, Guanghui He, and Haiyun Luo. 2006. Flow Scheduling for End-Host Multihoming.. In *INFOCOM*. Citeseer.

[29] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. 2011. Design, Implementation and Evaluation of Congestion Control for Multipath TCP.. In *NSDI*, Vol. 11. 8–8.