On Leveraging Multi-Path Transport in Mobile Networks

Yeon-sup Lim

ON LEVERAGING MULTI-PATH TRANSPORT IN MOBILE NETWORKS

A Dissertation Presented

by

YEON-SUP LIM

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2017

College of Information and Computer Sciences

ON LEVERAGING MULTI-PATH TRANSPORT IN MOBILE NETWORKS

A Dissertation Presented

by

YEON-SUP LIM

Approved as to style and content by:

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Professor Don Towsley for the consistent support, encouragement, and careful guidance throughout my doctorate study in UMass. Under his careful supervision, I have been able to seek interesting research topics and conduct the corresponding projects in this dissertation.

Besides my advisor, I would like to thank the rest of my thesis committee: Professor Deepak Ganesan, Professor Arun Venkataramani, Professor Lixin Gao, and Dr. Erich M. Nahum, for their insightful comments and encouragement.

I have also collaborated with colleagues from IBM Research and Cambridge University. I again would like to thank Erich M. Nahum; he has suggested a lot of ideas to enhance my proposed schemes and refine the experimental designs. Also, thank you to Richard J. Gibbens; his advices on data statistics have helped me to improve the analytical representations of experimental results.

This dissertation would not have been completed without the persistent support of my family and friends. With their support, I could withstand the tough times during my doctorate study. I am also grateful to Laurie Connors for her assistance to deal with my travel documents and experimental equipments orders. Last but not the least, thank you to Leeanne Leclerc for keeping my paperwork and records on track.

ABSTRACT

ON LEVERAGING MULTI-PATH TRANSPORT IN MOBILE NETWORKS

FEBRUARY 2017

YEON-SUP LIM

B.S., SEOUL NATIONAL UNIVERSITY

M.S., SEOUL NATIONAL UNIVERSITY

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Donald F. Towsley

Multi-Path TCP (MPTCP) is a new transport protocol that enables mobile devices to simultaneously use several physical paths through multiple network interfaces. MPTCP is particularly useful for mobile devices, which usually have multiple wireless interfaces such as IEEE 802.11 (WiFi), cellular (3G/LTE), and Bluetooth. However, applying MPTCP to mobile devices introduces new concerns since they operate in harsh environments with resource constraints due to intermittent path availability and limited power supply. The goal of this thesis is to resolve these problems so as to be able to practically deploy MPTCP in mobile devices.

The first part of the thesis develops a cross-layer path management approach that exploits information from the physical and medium access control layers to deal with intermittent path availability in mobile environment while increasing path utilization of MPTCP over lossy links. Experimental results show that this approach efficiently utilizes intermittently available WiFi paths, with throughput improvements of up to 72%.

In addition to the need to manage intermittent path availability, MPTCP must deal with the increased energy consumption due to simultaneous operation of multiple network interfaces. Hence, it is important to understand the energy consumption behavior to deploy MPTCP in mobile devices. We develop an energy model for MPTCP power consumption derived from experimental measurements. Experimental results show that the model accurately estimates the MPTCP energy consumption.

Based on the MPTCP energy model, we further explore conditions under which MPTCP is more energy-efficient than either standard TCP or MPTCP. Informed by this finding, we design and implement an energy-aware MPTCP variant for mobile devices. Experimental results show that our approach reduces power consumption compared to standard MPTCP by up to 80% for small file downloads and up to 15% for large file downloads, while preserving the availability and robustness benefits of MPTCP.

Finally, we study the impact of path asymmetry on MPTCP performance. Since path asymmetries frequently appear in mobile networks, it is crucial to design a MPTCP path scheduler that properly distributes traffic across available paths, which has different network characteristics such as round-trip time and bandwidth. We propose and implement a new MPTCP path scheduler, ECF (Earliest Completion First), that utilizes all relevant information about a path. Experimental results show that in the presence of path asymmetries ECF consistently utilizes all available paths more efficiently while mitigating out-of-order delay compared to the default scheduler.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

The deployment of mobile devices such as smartphones and tablets has significantly increased over the last few years. This rapid increase is based on the progress and ubiquity of wireless communication technologies, such as IEEE 802.11 (WiFi) and cellular (3G/LTE) networks. The advent of mobile devices with multiple wireless interfaces is leading to efforts to take advantage of these interfaces and simultaneously utilize them. Multi-Path TCP (MPTCP) is a new transport protocol being standardized by IETF to enables systems to exploit available paths through multiple network interfaces.

MPTCP is particularly useful for mobile devices, which usually have multiple wireless interfaces. The benefits of leveraging MPTCP in mobile devices are three-fold: First, by utilizing the available bandwidth of each subflow, an MPTCP connection can achieve higher throughput than a standard TCP connection [61]. Second, while connectivity in one network can degrade or disappear, MPTCP offers a seamless TCP connection by using paths (subflows) through another network [62]. Finally, the MPTCP layer is hidden from user applications by providing a standard TCP socket interface. Existing TCP applications need not to be modified to support MPTCP [20]. However, since mobile devices operate in harsh environments with limited resources, such as power, and where paths are intermittently available, deploying MPTCP on mobile devices introduces new concerns:

- Dynamic path availability in mobile networks

  Path quality frequently changes in mobile networks, for example, WiFi connectivity is often unusable in mobile scenarios due to the short signal ranges of access points (APs). However, MPTCP is unable to efficiently utilize frequently broken/recovered paths, as it relies on the use of a coarse-grained timeout mechanism.

- Additional energy overhead from operating multiple network interfaces

  MPTCP consumes additional energy to simultaneously operate multiple network interfaces. Since mobile devices are frequently constrained by the amount of power available in their batteries, judicious use of these interfaces is required.

- Path asymmetries in mobile networks

  Path asymmetries often appear in mobile networks, e.g., as a mobile device moves WiFi path can become unstable, consequently providing low bandwidth while a cellular path remains stable providing high available bandwidth. Such path asymmetries can result in inefficient use of the higher bandwidth path together with significant packet reorderings at the receiver.

1 Thesis Contribution

The main contributions of this thesis are:

- We develop a path management scheme for MPTCP to mitigate path underutilization problem that occurs when paths are intermittently connected. Our approach utilizes cross-layer information, such as physical bit rate and number of medium access layer frame retransmissions, to detect active/inactive path status so that MPTCP avoids unnecessary loss-related operations such as excessive exponential increases of retransmission timeout values, which results in inefficient use of recovered paths.

- We examine MPTCP energy consumption behavior via a combination of measurement, modeling, and experimentation. We measure power consumption of off-the-shelf mobile devices across a range of scenarios, varying download/upload size and available path bandwidth. We develop an MPTCP energy model derived from these experimental measurements.

- We design and implement a multi-path TCP variant for power-constrained mobile devices to improve the energy efficiency of MPTCP while having minimal impact on download latency. Our scheme is the first MPTCP implementation that monitors path characteristics at run time and dynamically chooses paths based on per-byte energy efficiency.

- We provide a thorough analysis of performance problems in MPTCP caused by path asymmetries when using the default scheduler. Using a streaming adaptive bit rate video workload, we demonstrate that MPTCP default scheduler does not achieve the ideal use of multiple paths in the presence of path asymmetries. Based on this insight, we design and implement a new path scheduler which takes path asymmetries into account.

2 Thesis Outline

Chapter 1 provides the background context for this thesis. Chapter 2 presents a cross-layer path management scheme, called MPTCP-MA that allows MPTCP to deal with dynamic path connectivity along with its implementation and evaluation in a mobile environment. Chapter 3 develops an MPTCP energy model based on experimental measurements taken on several mobile devices. Based on this energy model, in Chapter 4 we design and implement an improved energy efficient MPTCP, called eMPTCP, in actual mobile devices, and evaluate it in several scenarios. Chapter 5 reports on a performance degradation problem of the default MPTCP path scheduler when path asymmetries exist and presents a new MPTCP path scheduler to maximize path utilization, called ECF (Earliest Completion First). It concludes with an experimental evaluation of ECF. In Chapter 6, we summarize this thesis and discuss future research directions.

CHAPTER 1

MULTI-PATH TRANSPORT CONTROL PROTOCOL

MPTCP is a recently standardized TCP-compatible protocol, transparent to user applications, that takes advantage of multiple paths through several communication interfaces [20, 63, 76]. MPTCP allows a single data stream to be split across multiple paths referred to as subflows: subflows are defined logically by all end-to-end interface pairs. For example, if each host has two interfaces, an MPTCP connection consists of four subflows. These subflows are exposed to the application layer as one standard TCP connection, so that a user application needs not to be modified to utilize MPTCP. In MPTCP, each subflow processes packets in the same manner as standard TCP, except for congestion control. MPTCP uses a coupled congestion control algorithm across all subflows to guarantee TCP fairness at shared bottlenecks between standard TCP and MPTCP connections [76]. Since ordering is preserved within a subflow, but not across them, MPTCP must take care to combine subflows into the original ordered stream. To this end, MPTCP appends additional information called the data sequence number as a TCP header option to each packet. Based on the data sequence numbers, MPTCP merges packets from multiple subflows properly and delivers in-order streams at the connection level.

1.1 Connection Establishment

To initiate an MPTCP connection, each end host knows at least one of its peer's IP addresses. To establish an MPTCP connection, a client sends a connection request (SYN packet) with *MP-CAPABLE* option to the server's known IP address, which is similar to the connection establishment procedure of standard TCP. During this first SYN and SYNACK exchange, the client and server share a hash key to identify the MPTCP connection.

Once the connection request is sent, the client sends a packet with *Add Address* option if it has additional interfaces such as a 3G/LTE interface. After this address announcement, the client sends another SYN packet with a *JOIN* option using its announced address, together with the exchanged hash key for this MPTCP connection. If the address in *JOIN* differs from the address previously reported by *Add Address* option, the server updates the information about corresponding client address, e.g., the client announces a private address if the client is behind Network Address Translations (NATs), but the packet with *JOIN* option is sent with a public address beyond NATs. With this *JOIN* option, the subflow becomes associated with the current established MPTCP connection. If the server also has additional interfaces, the server sends an *Add Address* option to inform the client about the available address for the current MPTCP connection. As soon as the client receives it, the client sends out another connection request with *JOIN* option to the server's newly notified IP address and initiates a new subflow.

1.2 Congestion Controller

Similar to a standard TCP connection, each subflow in an MPTCP connection maintains its own congestion window. However, MPTCP can be unfair to normal TCP connections if it uses standard TCP congestion control algorithms such as Reno for each subflow. For example, assume that one normal TCP connection and one MPTCP connection share a single link and the MPTCP connection opens two subflows over the link. Then, the MPTCP connection takes twice more throughput over the link than the TCP connection, which is unfair.

In order to guarantee fair share of bandwidth with normal TCP connections, MPTCP uses a coupled congestion control referred to as Linked Increase Algorithm (LIA). Denote the congestion window and round trip time of subflow i by w_i and RTT_i. Let R be the set of all subflows in an MPTCP connection. LIA works as follows:

- For each ACK on subflow i: $w_i = w_i + \min\left(\frac{\max_{k\in R}\frac{w_k}{RTT_k^2}}{\left(\sum_{k\in R}\frac{w_k}{RTT_k}\right)^2}, \frac{1}{w_i}\right)$
- For each loss on subflow i: $w_i = \frac{w_i}{2}$

LIA limits congestion window increase by up to $1/w_i$ in order to be at most as aggressive as regular TCP. By linking the increase, LIA allocates more window to subflows with lower packet loss rates, i.e. less congested subflows, while it lets a set of subflows to take the aggregate bandwidth similar to the bandwidth that each competing TCP connection obtains.

Khalili et al. [37] shows that LIA fails to satisfy the design goals of MPTCP even though it provide better congestion balancing than Reno. As a revised linked control algorithm, they propose Opportunistic Linked Increase Algorithm (OLIA), which works as follow:

- For each ACK on subflow i: $w_i = w_i + \frac{\frac{w_i}{RTT_i^2}}{\left(\sum_{k\in R}\frac{w_k}{RTT_k}\right)^2} + \frac{\alpha_i}{w_i}$
- For each loss on subflow i: $w_i = \frac{w_i}{2}$

, where α_i is defined as:

$$\alpha_i = \begin{cases} \frac{1/|R|}{|B-M|} & \text{if } i \in B - M \neq \emptyset \\ -\frac{1/|R|}{|M|} & \text{if } i \in M \text{ and } B - M \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

, where M is the set of paths with the largest window sizes, B is the set of best paths in terms of packet loss rates.

If the paths have the largest window sizes, α becomes zero or negative, thus OLIA increases their windows slowly. If the best paths with lowest loss rates have small window sizes, OLIA quickly increases their windows by using a positive α. Hence, OLIA satisfies the design goals of MPTCP [21] and provides optimal load balancing [36].

1.3 Subflow Operation Modes

MPTCP has three modes of operation to control subflow usage. One of them is backup mode, where MPTCP opens TCP subflows over all interfaces, but uses only a subset of them for packet transmission [52]. If a user sets a particular interface to backup mode, MPTCP avoids sending traffic through the corresponding subflows. Then, in order to force a remote-side to avoid using the subflows, MPTCP sends a *MP_PRIO* on these subflows to the remote-side. However, MPTCP does not have any mechanism to control the mode of a subflow based on its connectivity status. In addition, the current backup mode mechanism is not able to properly handle the situation where an end host places an active interface into backup mode: e.g., MPTCP needs to carefully deal with in-flight packets (i.e., packets that are not acknowledged yet) on the subflows associated with such an interface.

1.4 Broken Paths in MPTCP

Standard TCP is unaware of whether a path is up or down: regardless of path status, a TCP sender retransmits lost packets and exponentially increases the retransmission timeout (RTO) for next retransmission when an acknowledgment does not arrive at the TCP sender within a RTO [59].

Similarly, MPTCP does not have a particular mechanism to manage path status changes. However, the current implementation [63] has an additional event handler to process address change events at a network interface:

- *NETDEV_UP*: This event occurs when an IP address is added to the interface. MPTCP then establishes the corresponding subflow associations.
- *NETDEV_DOWN*: This event is notified when the interface loses an IP address. With this event, MPTCP terminates all corresponding subflow connections.

Unfortunately, mobile devices can lose or regain paths without either of these events occurring in a mobile scenario. For example, a WiFi connection in a mobile device often becomes unusable due to the device moving outside the range of an access point, even when its WiFi interface still maintains an IP address.

1.5 Path Scheduler

When an MPTCP sender has data to send, it must choose a path over which to send that data. This is the task of the path scheduler. The default MPTCP path scheduler selects the subflow with the smallest RTT for which there is available congestion window (CWND) space for packet transmission.

However, this simple scheduling policy has a performance degradation problem due to bandwidth limitations caused by small receive windows when path asymmetry exists, e.g., one path has a lower bandwidth than others. To solve this problem, MPTCP has opportunistic retransmission and penalization mechanisms [53], which reinject unacknowledged packets from a slow path over a fast subflow and decrease CWND of the slow path.

CHAPTER 2

CROSS-LAYER APPROACH TO DEAL WITH DYNAMIC CONNECTIVITY IN MPTCP FOR MOBILE DEVICES

Wireless path characteristics change frequently in mobile environments, causing challenges for MPTCP: For example, WiFi associated paths often become unavailable as devices move, since WiFi provides intermittent connectivity due to the short signal range and susceptibility to interference. To maximize the benefit of multiple paths, it is necessary for MPTCP to not only utilize all available paths, but to determine path availability and quality as quickly as possible. However, MPTCP currently is unable to manage frequently broken/recovered paths, as it relies on the use of coarse-grained timeouts. This results in unnecessary delays after a path recovers, since the sender must wait until a successful retransmission occurs. This problem becomes more severe the longer a path is broken, as TCP's course-grained retransmission timeouts increase exponentially.

The goal of this chapter is to develop mechanisms that efficiently manage paths based on their connectivity status, in order to mitigate the delay of detecting and using restored paths in MPTCP. To this end, we propose MPTCP-MA, which controls path usage based on MAC-Layer information. MPTCP-MA exploits MAC-Layer information to estimate path status, and suspends/releases a path based on this estimation. By quickly detecting path failure and recovery, MPTCP-MA can avoid unnecessary losses and utilize recovered paths more quickly. We then experimentally evaluate an implementation of MPTCP-MA in a mobile environment.

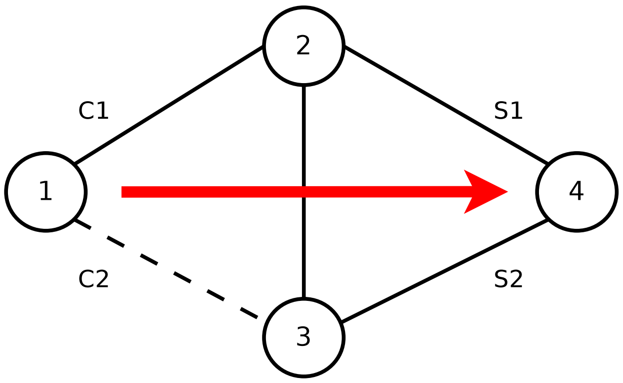


Figure 2.1. Simple Topology for MPTCP

2.1 Problem Motivation

In this Section we show how the problem of under-utilized paths is fundamental based on the way TCP responds to loss. We describe the utilization problem using both modeling and execution-driven simulation.

2.1.1 Under-utilization of Recovered Paths

In this subsection, we experimentally demonstrate the under-utilization problem that exists when a subflow recovers in MPTCP. Our experimental platform consists of the Common Open Research Emulator (CORE) [1] that allows us to control a link connectivity status in emulated networks, where each node runs MPTCP Linux kernel [63].

Consider the simple topology shown in Figure 2.1, where nodes 1 and 4 have two interfaces each, and node 1 is transmitting data to node 4 through an MPTCP connection. Assume that link $(1, 3)$ is a link that frequently breaks. Denote node 1's interfaces as $C1$ and $C2$ and node 4's interfaces as $S1$ and $S2$. In this case, MPTCP establishes four subflows denoted $C1 - S1$, $C1 - S2$, $C2 - S1$, and $C2 - S2$. Two subflows ($C2 - S1$ and $C2 - S2$) are affected by the status of link $(1, 3)$. In the rest of this capter, we use the following terminology for a subflow status: *active* - available for transferring packets, and *inactive* - unusable for packet transmissions. Note that

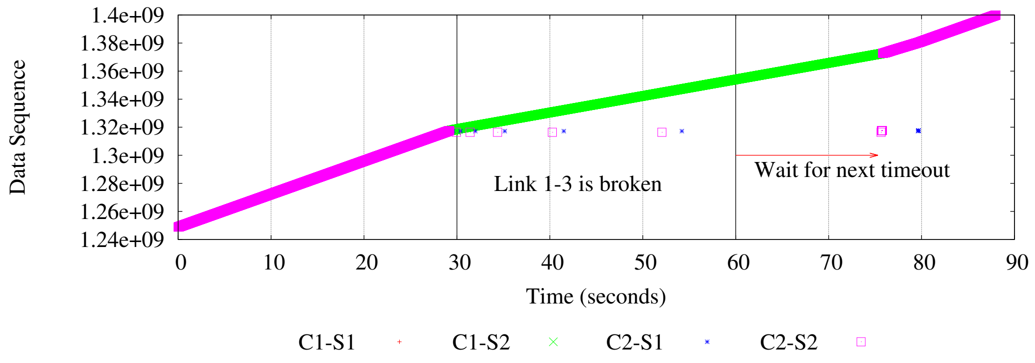


Figure 2.2. Re-use Delay of Active Subflow

we consider a scenario where inactive subflows are still valid (i.e., the interfaces have valid IP addresses) but become unusable, e.g., a weak WiFi association prevents communication but does not trigger an IP address change via a $NETDEV\_UP/DOWN$ event.

Figure 2.2 shows the TCP data sequence numbers over time when link $(1, 3)$ breaks at time 30 and recovers at time 60. The contributions of subflows $C1-S1$ and $C1-S2$ are obscured by those of other subflows even though packets are continuously transmitted through them. As can be seen by the change in slope from time 30 to time 75 in Figure 2.2, throughput is lower when subflows $C2 - S1$ and $C2 - S2$ are not used. We observe retransmissions based on exponential timeouts on the two subflows associated with link $(1, 3)$ (subflows $C2-S1$ and $C2-S2$) while it is broken. Even though link $(1, 3)$ recovers at time 60, the two subflows $C2 - S1$ and $C2 - S2$ are not used until their RTOs expire. Thus, MPTCP cannot utilize them for much time even though the subflows are active.

This behavior is illustrated in Figure 2.3 and modeled by a simple two state discrete time Markov chain where the state indicates the subflow status as active and inactive. Assume that a packet transmission (data and ACK exchange) completes in one unit time and a node can perfectly measure RTTs. Denote the probability that

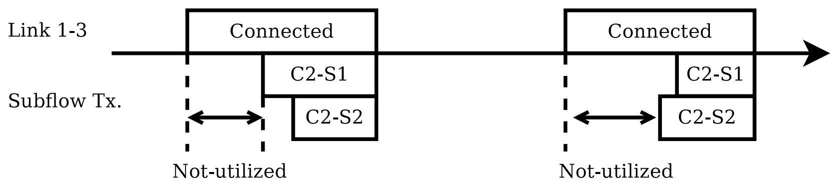


Figure 2.3. Timing Diagram of Retransmission

a subflow becomes inactive by λ_i and the probability that a subflow becomes active by λ_a. The expected inactive and active durations after they start, $E[D_i]$ and $E[D_a]$, can be calculated as:

$$\begin{aligned} E[D_i] &= \sum_{k=1}^{\infty} k(1-\lambda_a)^{k-1}\lambda_a = \frac{1}{\lambda_a} \\ E[D_a] &= \frac{1}{\lambda_i} \end{aligned}$$

In this model we assume an RTT of one when a packet transmission is successful, and the RTO value for the nth timeout to be 2^n. Suppose that one RTO occurs at the start of an inactive period. Then, the expected number of subsequent RTOs until a subflow becomes active, $E[N]$, can be approximated as

$$E[N] \approx \log_2 \frac{1}{2}\left(E[D_i]+1\right) = \log_2 \frac{1}{2}\left(\frac{1}{\lambda_a}+1\right).$$

The subflow can become active in the middle of an RTO. We approximate the average remaining duration of this RTO, $E[R]$, as

$$\begin{aligned} E[R] &\approx \sum_{k=0}^{2^{E[N]+1}} \left(2^{E[N]+1}-k\right)(1-\lambda_a)^{2^{E[N]+1}-k}\lambda_a, \\ &= \frac{1-\lambda_a-(2+\lambda_a)(1-\lambda_a)^{\frac{1}{\lambda_a}+2}}{\lambda_a}. \end{aligned}$$

Thus, the fraction of unutilized active subflow period F, is approximated as

$$\begin{aligned} F &\approx \min\left(\frac{E[R]}{E[D_a]}, 1.0\right), \\ &= \min\left(\frac{\lambda_i(1-\lambda_a-(2+\lambda_a)(1-\lambda_a)^{\frac{1}{\lambda_a}+2})}{\lambda_a}, 1.0\right). \end{aligned}$$

Figure 2.4 presents F as a function of λ_i for different values of λ_a. As shown in Figure 2.4, the active period is utilized less as it becomes shorter, i.e., larger λ_i. With a particular expected length of active period, longer inactive period, i.e., smaller λ_a, results in lower utilization of active period.

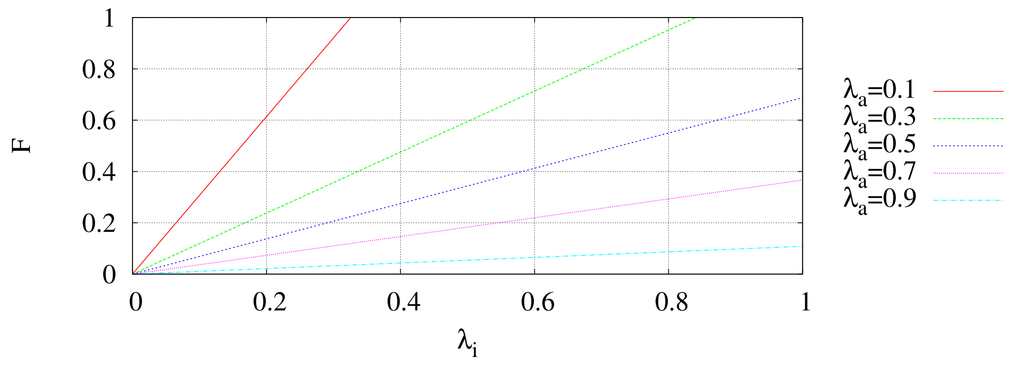


Figure 2.4. F according to λ_i and λ_a

We perform further experiments using CORE to estimate the fraction of the unutilized active period as a function of the inactive duration. Assume that link $(1, 3)$ in Figure 2.1 is connected for 30 seconds and then breaks for a time of duration X, after which it is again connected for another 30 seconds. While repeating this procedure ten times, we investigate the fraction of time during which subflows $(C2 - S1$ and $C2 - S2)$ are unutilized even though they are active, i.e., link $(1, 3)$ becomes connected.

Figure 2.5 presents the fraction of unutilized active period averaged over the two subflows $C2 - S1$ and $C2 - S2$ as a function of the inactive duration X. As shown in the figure, the subflows associated with link $(1, 3)$ are not utilized as fraction of the inactive duration X increases: the case of a 30 sec disconnection is an exception because link recovery time is close to a timeout expiration, compared to the case of 40 sec disconnection. We also observe that the experimental results accord closely to predictions made by the model in spite of the many assumptions. Based on both the experiments and the model, we see that a subflow is not efficiently utilized, as the inactive duration becomes longer than the active duration, since it has to wait for the next successful retransmission following a timeout.

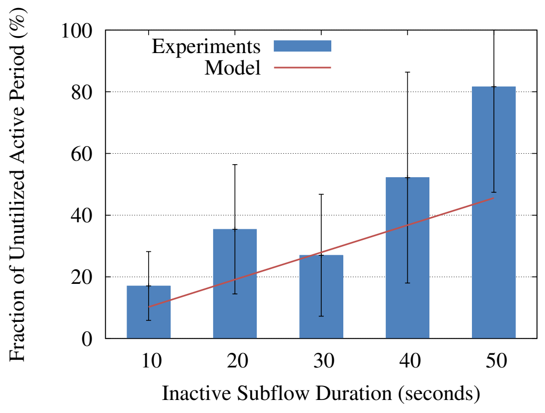


Figure 2.5. Unutilized Active Subflow Period

2.2 Path Management Approach

In this section, we describe our approach to solving the subflow under-utilization problem when subflows are restored from an inactive status, by incorporating with MAC-Layer awareness; the resulting protocol is MPTCP-MA. Using WiFi as an example, MPTCP-MA includes a simple yet effective approach to estimate a subflow status using MAC-Layer information and corresponding subflow management to support efficient use of active subflows.

2.2.1 Motivation

The IEEE 802.11 standard defines receiver sensitivity, which is the minimum signal strength for a receiver to be able to acceptably decode frames [66]. Receiver sensitivity is governed by the underlying receiver noise floor, which is a measure of interference from all noise sources and unwanted signals: as the noise floor increases, receiver sensitivity increases. Since it is difficult to measure the actual noise floor at a mobile device, we cannot determine the exact required receiver sensitivity.

To develop a method to determine the threshold of receiver sensitivity for estimating subflow status, we first investigate MAC-Layer behavior with MPTCP according to WiFi signal strength. To this end, we perform a set of experiments in a static scenario. After setting up an 802.11g access point at a fixed location, we locate a mobile device at varying distances from the AP as a way to control signal strength. Since signal strength fluctuates over time, we choose the location for each signal strength range after measuring signal strength for one minute. Then, the mobile device uploads a 30 MB file to the server five times at each location.

Figure 2.6 presents the average MPTCP throughput as a function of the measured signal strength. As shown in Figure 2.6, MPTCP throughput over WiFi decreases as WiFi signal strength becomes weaker: in particular, WiFi throughput is almost zero when the signal strength falls below -71 dBm. We see that -71 dBm is the receiver sensitivity which can be used as a signal strength threshold to determine when a WiFi subflow becomes inactive. However, this threshold can vary due to environmental factors such as underlying noise, thus, we need an adaptive method to determine a proper estimate.

We investigated MAC-Layer frame transmission in the experiments and observe that MPTCP still directs packets to WiFi even when the signal strength falls below -71 dBm, but the MAC-Layer frequently fails to deliver them to the AP after multiple frame transmission retries. Figure 2.7(a) shows the ratio of the number of MAC-

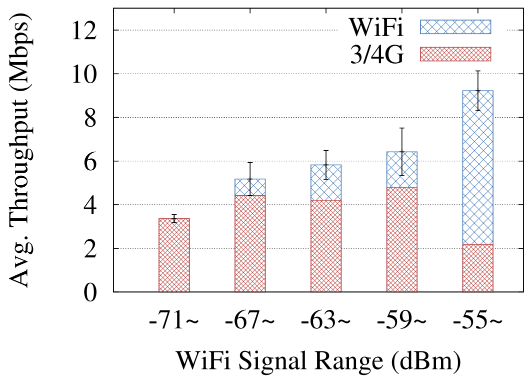
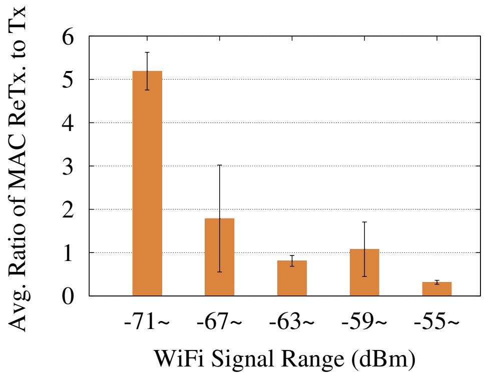
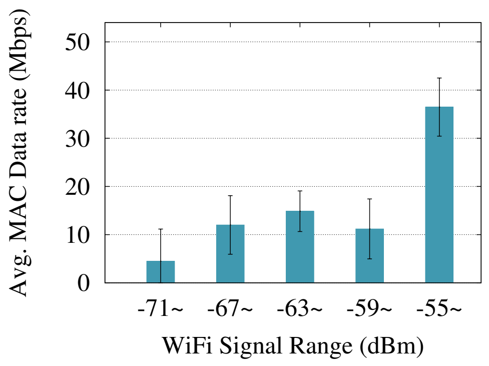


Figure 2.6. Average MPTCP Uplink Throughput according to Signal Strength



(a) Ratio of Mac-Layer ReTx. to MAC-Layer Tx.



(b) Average Tx. Data Rate

Figure 2.7. MAC-Layer Behavior according to Signal Strength

Layer frame retransmissions to the number of transmissions versus signal strength. Throughput decreases, and the MAC layer performs more retransmissions than transmissions, as the signal strength becomes weaker. In particular, WiFi throughput approaches zero when retransmission ratio is larger than one. This indicates that the ratio of frame retransmissions can be used to to determine the WiFi signal strength threshold for the subflow status estimation.

Rate adaptation is one of the basic functionalities in IEEE 802.11, which is designed to cope with the variation of wireless channels by exploiting the multi-rate

	Definition
$R(t)$	Measured PHY Data Rate at Unit Time t
R_{min}	Lowest PHY Data Rate
$N_{tx}(t)$	Number of Frame Transmissions at Unit Time t
$N_{retx}(t)$	Number of Frame Retransmissions at Unit Time t
$S(t)$	Measured Signal Strength at Unit Time t
T_I	Signal Strength below which WiFi subflow is labeled inactive
T_A	Signal Strength above which WiFi subflow is labeled active
α	Sensitivity Parameter
β	Parameter to determine T_A from T_I
γ	Polling Interval to estimate path quality

Table 2.1. Notations

capability provided by the 802.11 physical layer (PHY) [8]. Figure 2.7(b) presents the average measured PHY data rate vs. WiFi signal strength. Note that the maximum data rate is 54 Mbps since we use an 802.11g AP for the experiments. As shown in Figure 2.7(b), the WiFi PHY reduces the data rate as the signal strength becomes weaker. Recall that the ratio of frame retransmissions in Figure 2.7(a) becomes greater than one when the WiFi signal strength is less than -71 dBm. This means that the MAC layer experiences more retransmissions than transmissions, even with a low PHY data rate, if the WiFi link has poor quality. In the following subsection, we present a method for estimating WiFi subflow status inspired by these findings.

2.2.2 WiFi Subflow Status Inference

Here, we present our method for inferring the status of a subflow associated with WiFi using available MAC-Layer information. The idea behind our method is that, when a WiFi link becomes unavailable (i.e., the subflow using that WiFi is inactive), the number of MAC-Layer frame retransmissions becomes larger than the number of successful frame transmissions even at the lowest PHY data rate. We use this information to explicitly notify MPTCP that a subflow is inactive. In contrast to

Algorithm 1 WiFi Subflow Status Inference

```
For each WiFi subflow

for every $\gamma$ ms do
    if $R(t) = R(t-1) = R_{min}$ then
        if $N_{retx}(t) - N_{retx}(t-1) \geq \alpha(N_{tx}(t) - N_{tx}(t-1))$ then
            $T_I = S(t)$
            $T_A = (1-\beta)T_I$ /* $T_I$ is negative */
        end if
    end if
    if $S(t) \leq T_I$ then
        Label WiFi subflow as Inactive
    else if $S(t) \geq T_A$ then
        Label WiFi subflow as Active
    end if
end for
```

previous approaches [29] which implement an explicit link failure notification (ELFN) using a detection of routing failure, e.g., "host unreachable" response from an ICMP message. Our approach locally infers subflow status without help from other nodes such as routers. Notably, ELFN's approach of relying on an ICMP "host unreachable" message cannot work in our setting since the association of our device to the AP remains valid and the AP, therefore, cannot reply with "host unreachable".

Table 2.1 and Algorithm 1 present the notations and inference method, respectively. The algorithm determines the status of a particular WiFi subflow using a signal strength threshold T_I determined from MAC-Layer information. T_I is set to the signal strength at which a mobile device experiences a frame retransmission ratio larger than α, $0 < \alpha < 1$, with the lowest PHY data rate during a particular interval.

Our algorithm uses a second signal strength threshold, $T_A = (1-\beta)T_I$, $0 < \beta < 1$, for switching the inactive status of WiFi subflow to active. Once the subflow is labeled inactive, it remains labeled as such until the signal strength rises above T_A even if it rises above T_I. This adds some hysteresis to the system and prevents it from switching states too frequently.

We use this approach for both uplink and downlink. Both ACK and data packets are counted at the MAC-Layer, thus, our algorithm can determine T_I reflecting the WiFi subflow status when sending ACKs. However, note that wireless link status is often asymmetric due to several factors such as heterogenous hardware, interference variation, and antenna diversity [46, 35]: for example, the uplink for ACKs can be still be usable even when the downlink for data is not available. In addition, ACK packets are more robust to wireless bit errors than data packets due to their smaller size: packet error rate decreases as packet size decreases [79]. This can degrade the accuracy of our algorithm. In Section 2.4, we will compare the performance of our algorithm for the uplink and downlink cases.

2.2.3 Discussion about Status Estimation of 3G/LTE subflows

Our estimation approach in MPTCP-MA can be extended to support subflows associated with other wireless interfaces, such as 3G/LTE. MPTCP-MA introduces a subflow quality measure similar to routing metrics, such as ETX [13], in order for MPTCP to manage subflow usage. Whereas routing-based approaches depend on packet information exchanges designed for routing protocols, MPTCP-MA estimates a subflow quality measure solely based on locally available information (namely, MAC-Layer information). Therefore, the idea underlying our subflow status inference is applicable to any kind of wireless network interface provided the required information is available.

3G/LTE MAC-Layer also performs link-layer retransmissions based on an ARQ-like scheme and changes PHY data rate using several modulation schemes [2, 9]. Current operating systems for mobile devices, such as iOS and Android, provide information about 3G/LTE signal strength. Thus, our estimation approach for 3G/LTE interfaces can be implemented in MPTCP-MA by utilizing that information. However, in this chapter, we focus on the implementation and evaluation for WiFi, since

3/4G network connectivity is widely available, while WiFi exhibits more intermittent connectivity.

2.2.4 Handling WiFi Subflow according to Estimated Status

Since MPTCP in mobile devices can utilize multiple paths using WiFi and 3/4G interfaces, an MPTCP connection can maintain an established state even when it disables WiFi subflows. This enables MPTCP-MA to suspend an inactive WiFi subflow similar to [23, 29] without losing a valid TCP connection. MPTCP-MA deals with a WiFi subflow labeled inactive by placing it into backup mode, cancelling its current retransmission timer, and moving in-flight packets to other subflows.

Note that existing approaches [23, 29] need to periodically probe a suspended TCP flow in order to check the availability of the flow. For example, in [29], a sender sends an unacknowledged packet with the lowest sequence number every two seconds to probe the suspended flow and waits for a response. In contrast, MPTCP-MA enables the subflow to transmit packets as soon as the local inference method decides that a suspended subflow is active.

When MPTCP-MA decides that the status of subflow has changed (either inactive or active), this decision needs to be communicated to the remote-side, particularly when it is a sender. To inform the remote side of the state change, MPTCP-MA introduces a new MPTCP option MP_SST, which is similar to MP_PRIO but triggers the suspending/releasing procedure, such as moving in-flight packets. This is added to a packet on a subflow through a 3G/LTE subflow. If one of the end hosts receives an MP_SST option from the other end host, it updates the state of the corresponding subflow (associated with the WiFi interface on the receiver), suspending or releasing the subflow, and moving in-flight packets to another path if necessary.

2.3 Implementation of MPTCP-MA

2.3.1 MPTCP Porting onto Android Devices

We ported the Linux MPTCP Kernel onto Android running on a Samsung Galaxy S3 for AT&T (SGH-I747) using a customized Jellybean 4.1.2 platform (CM10) with a Kernel 3.0.2 [12]. In addition to porting to a specific kernel, the platform has been modified to support the following features of MPTCP:

- First, we modified the Android connectivity manager to support simultaneous use of multiple interfaces. In Android, the current interface is taken down if a more preferred interface (e.g., WiFi) becomes available. We disabled this default behavior, instead adding the connectivity to a set of available interfaces.

- Second, we have inserted a policy routing feature based on a source address to the Android network daemon. The MPTCP kernel cannot appropriately use available paths until the routing information based on source addresses is correctly configured. With the modified network daemon, the device automatically sets up such required routing entries for MPTCP even in mobile scenarios where routing information changes frequently.

2.3.2 MAC-Layer Awareness for MPTCP

First, we explain our implementation for adding WiFi MAC-Layer awareness to MPTCP. To this end, we employ a Subflow Status Tracker (SST) that helps each MPTCP subflow access MAC-Layer information such as measured signal strength and number of MAC-Layer frame transmissions/retransmissions. Once MPTCP adds a socket for a new subflow, SST is set up for the socket and registered into the workqueue (`mptcp_sst_wq`) that is created when MPTCP-MA starts. The following code describes the SST initialization step at the Kernel: SST for the added TCP socket is initialized with the function `mptcp_sst_worker` in which SST functionalities are implemented, and it is enqueued into the workqueue `mptcp_sst_wq`. Then, SST continues to monitor the MAC-Layer information of the WiFi interface every γ ms if the socket is destined for a WiFi interface.

```
int mptcp_add_subflow(struct mptcp_cb *mpcb,
            struct tcp_sock *tp, gfp_t flags) {
    ....
    INIT_WORK(&tp->sst_work, mptcp_sst_worker);
    if(mptcp_sst_wq)
        queue_work(mptcp_sst_wq, &tp->sst_work);
    ....
}
```

To check if the destination of socket is a WiFi interface, SST first obtains the destination entry of the socket by using `sk_dst_get()`. The structure variable for destination entry (`struct dst_entry`) includes a pointer to the related network device (`struct net_device *dev`), which has a pointer to wireless device information (`struct wireless_dev *ieee80211_ptr`). Thus, SST can identify whether a socket is connected to a WiFi interface or not by checking if `dev` has a valid `ieee80211_ptr`.

SST retrieves the MAC-Layer information of the WiFi interface by a function in `struct cfg80211_ops` of `struct cfg80211_registered_device`: After SST converts the `ieee80211_ptr` to a `struct cfg80211_registered_device` variable (`rdev`), it calls `rdev->ops->get_station()`, which is linked to a driver-specific function that provides MAC-Layer information required for our inference method such as signal strength and number of MAC-Layer transmission retries.

The following pseudo code encodes the procedure for obtaining MAC-Layer information from a socket `sk`. Note that the available MAC-Layer information varies from driver to driver implementation. Since the WiFi interface driver of our device does not report the number of MAC-Layer transmissions and retransmissions through `get_station()`, we modify the linked function in the driver implementation to provide them.

```
struct dst_entry *dst = sk_dst_get(sk);
if(dst->dev->ieee80211_ptr) { /* destined to WiFi */
    ....
    /* rdev is converted from dst->dev->ieee8011_ptr */
    rdev->ops->get_station(info);
    ....
}
```

In case of other interfaces such as 3G/LTE, we can access their network statistics by `dev->netdev_ops->ndo_get_stats()`, which provides the general (not WiFi-specific) information such as the number of transmitted packets and the number of transmission errors.

2.4 Evaluation

2.4.1 Experimental Setup

Our experimental setup consists of a wired server equipped with an Intel Quad Core I7-3770 CPU and 32 GB of memory, a 802.11g WiFi access point, and a mobile device (Samsung Galaxy S3 SGH-I747).

The Linux MPTCP kernel [63] including MPTCP-MA is ported to our mobile device running a customized Jellybean 4.1.2 platform using a 3.0.2 kernel. The WiFi driver implementation in the kernel is also modified to provide all the required MAC-Layer information through function calls accessible to the TCP-Layer.

The server is connected through a single Gigabit Ethernet interface to our campus network and runs Ubuntu Linux 12.04 with our MPTCP-MA implementation. The mobile device is connected to the Internet using both 3G/LTE from AT&T and WiFi.

The default value of T_I is set to -80 dBm since it is the reference receiver sensitivity of IEEE 802.11 standard at an 1 Mbps data rate [32]. We set the frame retransmission ratio threshold α to one, because our experiments in Section 2.2 show that there is no WiFi throughput gain when the frame retransmission ratio is larger than one. We set $\beta = 1/16$ to provide a sufficient margin for T_A. The polling interval γ is set to

300 ms, so that the MAC-Layer observes a sufficient number of packet transmission trials (around 200 trials even at a 1 Mbps data rate). While these values work well in practice, it is possible that better results could be achieved using optimized values. Refining these values to improve performance remains as future work.

2.4.2 Mobile Scenario

To investigate the performance of MPTCP-MA in an actual mobile scenario, we measure bandwidth and other metrics while moving along the route shown in Figure 2.8, with the mobile device either downloading from or uploading to our server. The device is sometimes within WiFi communication range, and sometimes outside it, depending on its location. To make our comparison between MPTCP and MPTCP-MA as fair as possible, we use as similar routes as we can for the experiments. Using this mobile scenario, comparison metrics are calculated by averaging over the results from five experiments, where one experiment consists of three round trips along the route.

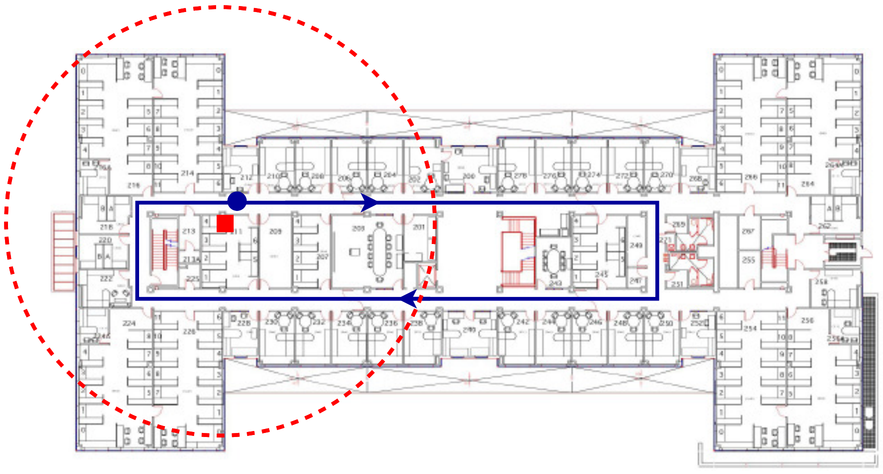


Figure 2.8. Mobile Scenario inside our department building (Start from the blue point. AP is located at the red point. The red dashed circle is the estimated usable access range of AP)

Interface	Uplink	Downlink
WiFi	48.07 (±15.63)	112.60 (±56.12)
LTE	182.83 (±8.53)	69.02 (±4.60)
3G	713.15 (±59.30)	175.24(±18.02)

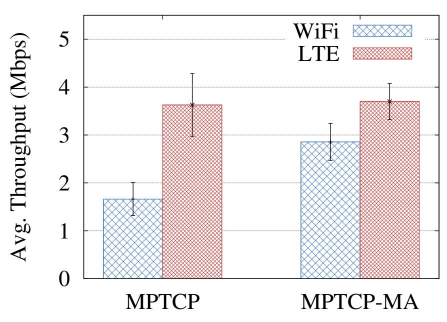
Table 2.2. Round Trip Time (ms)

Table 2.2 lists the measured average RTTs of subflows in our traces through each interface. Note that exceptionally large WiFi RTTs due to inactive periods are excluded when calculating the average. Therefore, the table entities are averages when subflows are available for transferring packets. We observe that the LTE connection status is quite stable during our experiments, i.e., RTT variance is small. In contrast, the WiFi connection exhibits RTTs with a larger standard deviation than LTE due to mobility, even though we only consider RTTs during the active periods. In the case of 3G, the average uplink RTT is much higher than WiFi and LTE, but the average downlink RTT is comparable to WiFi's.

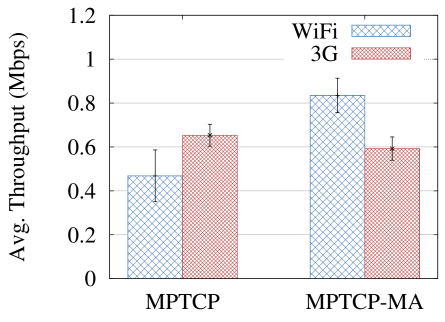
2.4.3 Uplink Experiments

We first present results for the uplink scenario. In this case, since the mobile device is the sender, MPTCP-MA determines the WiFi subflow status directly from the WiFi interface. This allows us to determine how effectively MPTCP-MA works with intermittent WiFi connectivity.

Figure 2.9 shows the average uplink throughput of each interface for both MPTCP and MPTCP-MA. We observe that MPTCP-MA achieves higher aggregate throughput than MPTCP. Upon closer examination, we see that the 3G/LTE component of throughput is similar in both cases, as expected. In contrast, the average WiFi throughput of MPTCP-MA increases over that of MPTCP by about 72% (with LTE) and 78% (with 3G). This demonstrates that MPTCP-MA exploits the available WiFi uplink bandwidth more effectively than MPTCP (Note that the gain obtained using MPTCP-MA differs according to the mobile scenario: If the inactive period is short



(a) WiFi and LTE



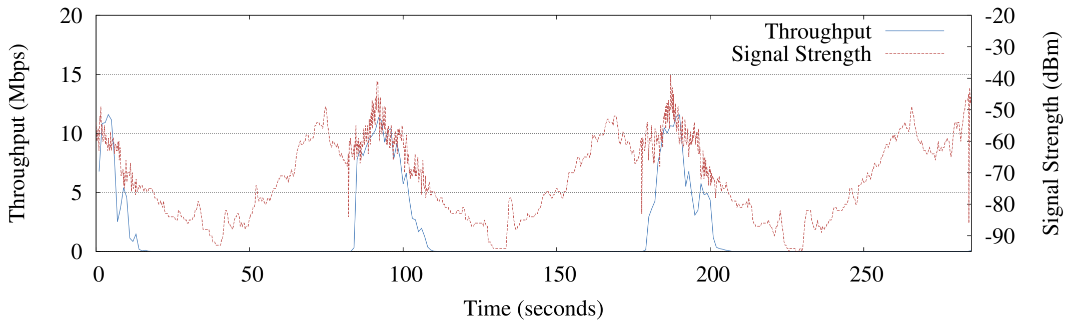
(b) WiFi and 3G

Figure 2.9. Average Uplink Throughput

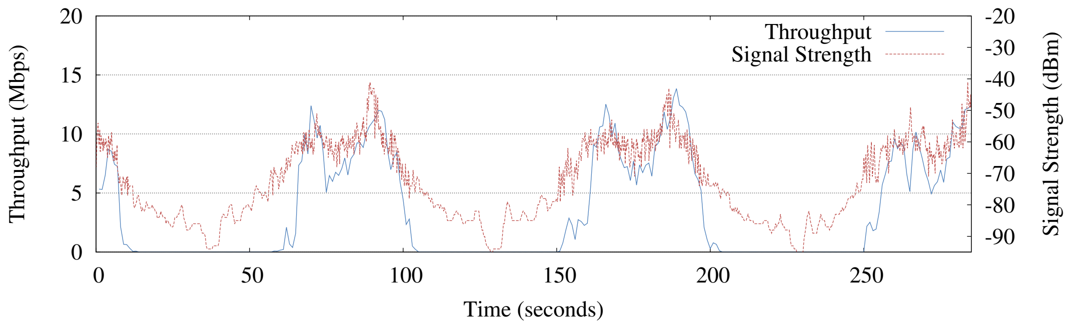
enough not to trigger many timeouts, MPTCP might be able to obtain performance similar to MPTCP-MA).

Figure 2.10 shows an experiment expressed as a time series where the two MPTCPs use LTE with WiFi. The figure presents the WiFi throughput as a function of time when each MPTCP uses the LTE interface together with WiFi, in blue, on the left Y-axis. The figure also shows the WiFi signal strength, in red, on the right Y-axis. Here, we select a sample trace for each MPTCP to illustrate the advantage of MPTCP-MA. Experimental conditions such as WiFi channel status and route paths cannot be exactly the same across experiments, however, the WiFi signal strength trace shows that the experiments for both configurations exhibit similar signal strength. In this experiment, the WiFi subflow is active during the interval $(50, 100)$, $(150, 200)$, and $(250, 300)$. We observe that MPTCP-MA quickly starts using the active WiFi subflow.

Figure 2.11 shows CWND values for the WiFi subflow over time. As shown in Figure 2.11, MPTCP-MA does not prevent all timeouts since it manages subflow usage every γ ms (300 ms): when packets starts to be transmitted through the WiFi subflow, CWND of MPTCP-MA increases from one as in MPTCP. However, MPTCP-MA starts increasing CWND earlier than MPTCP by preventing unnecessary successive



(a) MPTCP



(b) MPTCP-MA

Figure 2.10. Uplink WiFi Throughput over Time when using WiFi and LTE

timeouts. In our experiments, the mobile device moves towards the AP once WiFi becomes usable. We see that the MPTCP-MA CWND fluctuates until the signal strength becomes sufficiently strong, while the MPTCP CWND increases without oscillation since MPTCP tries to use the WiFi subflow *only after* the mobile device is close enough to the AP, missing part of the period where the WiFi subflow is active.

We also see the benefit of MPTCP-MA from the RTTs of packets transferred through WiFi subflow. Figure 2.12 presents measured RTTs when a packet is acknowledged through the WiFi subflow in the trace shown in Figure 2.10. As shown in Figure 2.12, packets transmitted right before the WiFi breaks experience RTTs almost equal to the length of inactive period, e.g., at around time 50, a packet is acknowledged with a measured RTT of around 40 seconds, which is similar to the length of the inactive period $(10, 50)$. Note that the MPTCP subflow scheduler chooses a subflow

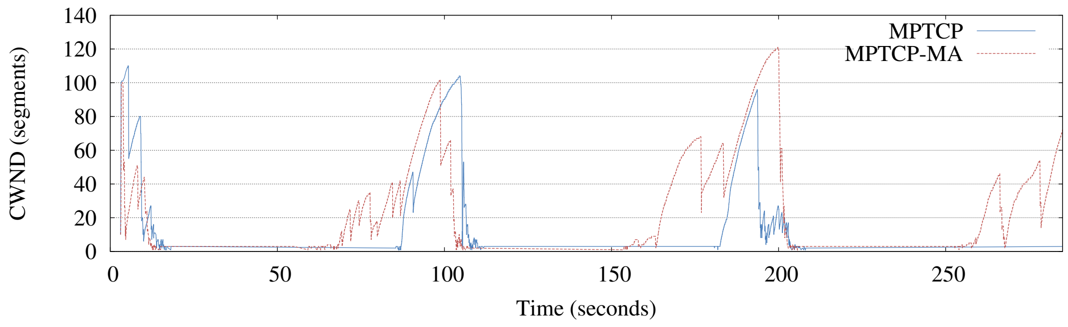


Figure 2.11. Uplink WiFi Subflow CWND over Time using WiFi and LTE

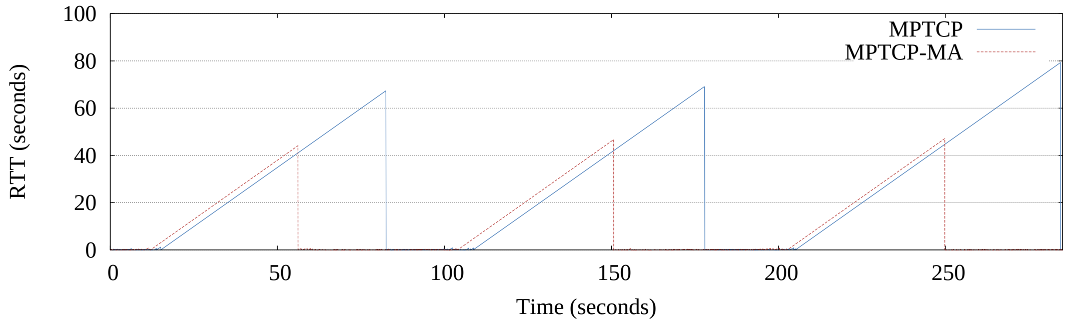


Figure 2.12. Uplink RTTs through WiFi subflow when using WiFi and LTE

for packet transmission that is not in backup mode, has a sufficiently large CWND, and has the smallest RTT [63]. By suspending an unusable subflow, MPTCP-MA avoids over-estimating RTTs of the subflow, quickly converging to an estimated RTT close to the real value of the subflow after it recovers. In our case, an intermittently active subflow is a WiFi subflow that has a smaller average uplink RTT (48.07 ms) than LTE (182.83 ms). Therefore, the default subflow scheduler is more likely to select an active WiFi subflow to transfer packets, that is, MPTCP-MA can more aggressively utilize an active WiFi subflow than MPTCP.

Next, we examine the timeout behavior of each type of MPTCP, which can affect the performance when a WiFi subflow becomes active. Table 2.3 presents the average number of timeouts and average peak RTO values. We observe that MPTCP-MA

	Avg. # of Timeouts		Avg. Peak RTO (ms)	
	with LTE	with 3G	with LTE	with 3G
MPTCP	49.2 (±10.5)	34.6 (±6.0)	1982.0 (±912.9)	2772.9 (±803.0)
MPTCP-MA	40.0 (±4.4)	33.2 (±4.7)	180.3 (±30.3)	210.1 (±44.0)

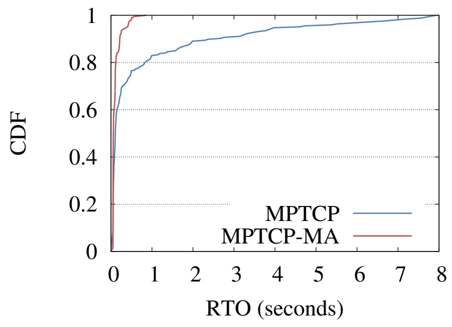
Table 2.3. WiFi Uplink TCP-Layer Timeout Behavior

exhibits slightly fewer timeouts than MPTCP regardless of which cellular technology is used. However, the average peak RTO value of MPTCP-MA is much smaller than that of MPTCP. We see that MPTCP-MA successfully avoids continuous timeouts during periods when the WiFi subflow is inactive. This result explains why MPTCP-MA obtains better WiFi throughput than MPTCP in Figure 2.9: MPTCP is likely to wait for a long time to achieve a successful retransmission and start utilizing an active WiFi subflow, whereas MPTCP-MA utilizes it much more quickly.

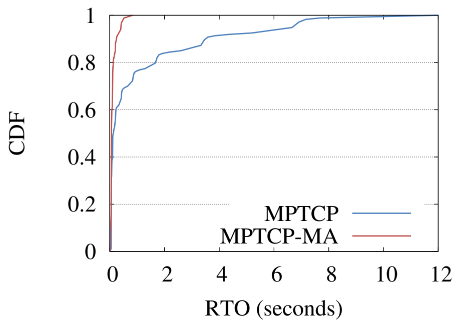
Figure 2.13 presents the cumulative distribution function (CDF) of RTO values for the WiFi subflow with the two types of MPTCP. As expected from Table 2.3, MPTCP exhibits large RTO values sometimes exceeding five seconds while MPTCP-MA exhibits RTO values always less than two seconds. Again, this is because MPTCP-MA halts retransmissions once it decides that a WiFi subflow becomes inactive. Therefore, we expect MPTCP-MA to obtain a successful retransmission earlier than MPTCP, even when MPTCP-MA suffers a timeout while it starts packet transmissions in the middle of path recovery.

2.4.4 Downlink Experiments

Next, we compare MPTCP and MPTCP-MA when the mobile device moves while downloading a file from the server. Since the interface of the sender (the server) is a wired Ethernet interface, MPTCP-MA cannot estimate the status of a WiFi subflow at the sender-side; only the receiver can locally estimate its WiFi subflow

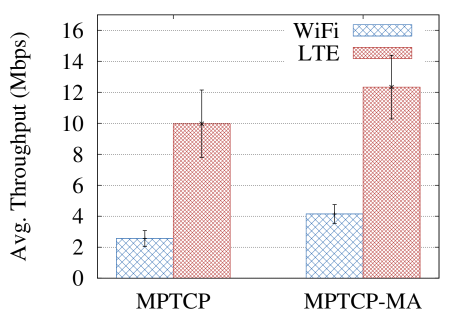


(a) WiFi and LTE

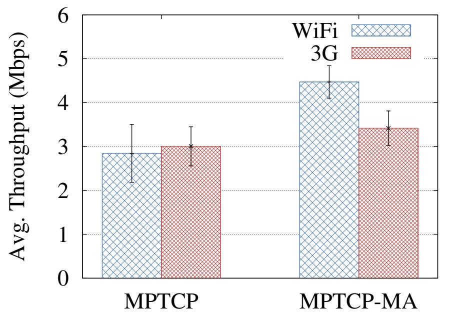


(b) WiFi and 3G

Figure 2.13. CDF of RTO values for WiFi subflow (Uplink)



(a) WiFi and LTE

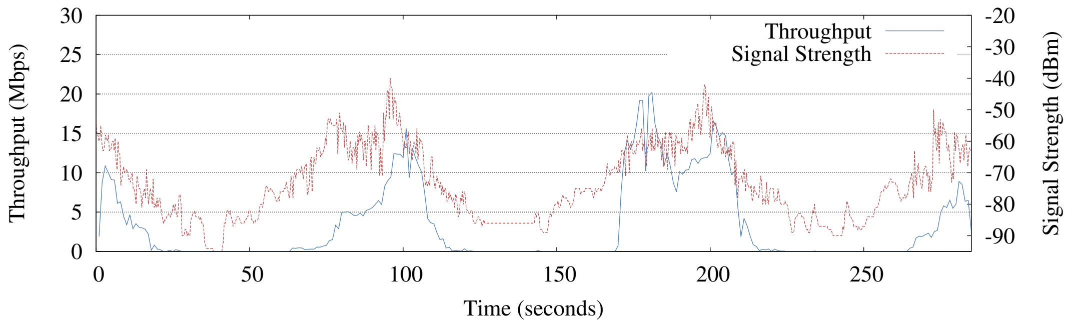


(b) WiFi and 3G

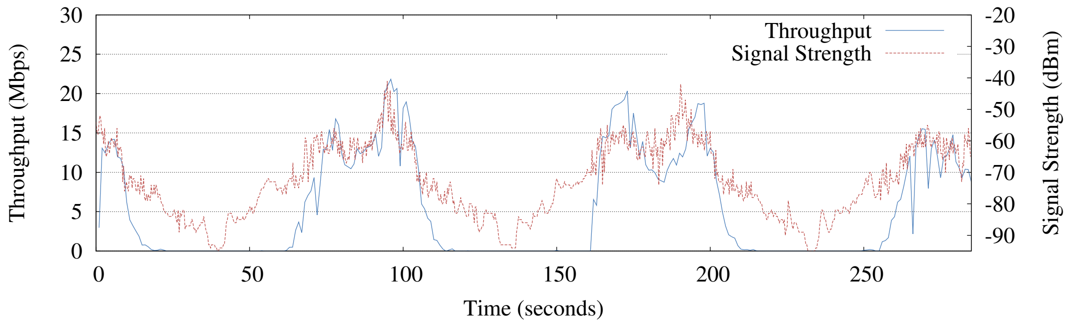
Figure 2.14. Average Downlink Throughput

status. Based on the status of the WiFi subflow indicated by the MP_SST option, the MPTCP-MA sender can control its behavior, while the MPTCP sender cannot.

Figure 2.14 presents the average downlink throughput of each interface for both MPTCP protocols using WiFi/LTE and WiFi/3G. We observe that MPTCP-MA yields better performance than MPTCP, achieving higher throughputs through both interfaces. As illustrated in Figure 2.14, the average MPTCP-MA WiFi throughput increases about 62% (with LTE) and 57% (with 3G) relative to MPTCP. This indicates that MPTCP-MA can utilize active WiFi subflows more quickly than MPTCP, even in the case of a downlink. The WiFi throughput improvement in the downlink scenario is smaller than that shown in the uplink experiments, but note two distinct



(a) MPTCP



(b) MPTCP-MA

Figure 2.15. Downlink WiFi Throughput over Time when using WiFi and LTE

differences. First, LTE download bandwidth is greater than 10 Mbps rather than around 4 Mbps in the uplink scenario. Second, a delay to notify the other end of the MPTCP connection is in this case roughly 69 ms over the LTE connection and 175 ms over the 3G connection. Despite these disadvantages, we still see improvement with MPTCP-MA.

Figure 2.15 shows the WiFi downlink throughput of each MPTCP using WiFi and LTE from selected traces as a function of time. Similar to the uplink experiments, MPTCP-MA more effectively uses the active period of the WiFi subflow. In particular, MPTCP-MA efficiently utilizes the second and fourth available periods while MPTCP loses part of these periods. We also observe that the behavior of CWND and RTT supports the gain of MPTCP-MA consistent with Figures 2.11 and 2.12; however, we omit those figures due to space limitations.

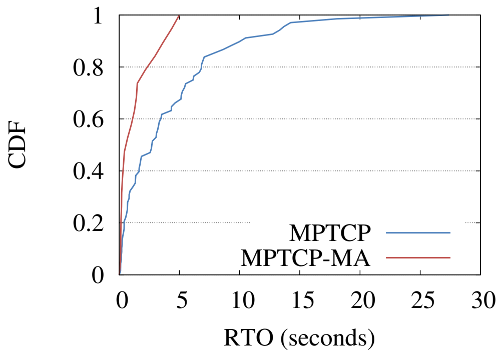
	Avg. # of Timeouts		Avg. Peak RTO (ms)	
	with LTE	with 3G	with LTE	with 3G
MPTCP	13.6 (±7.8)	11.8 (±5.2)	10295.6 (±5580.0)	6889.5 (±1723.9)
MPTCP-MA	3.8 (±1.3)	4.8 (±1.7)	2482.2 (±1549.6)	5441.0 (±3353.7)

Table 2.4. WiFi Downlink TCP-Layer Timeout Behavior

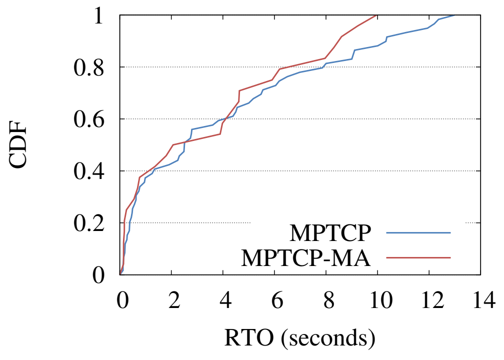
Table 2.4 shows the average number of timeouts and average peak RTO values in the downlink experiments. Both versions of MPTCP suffer smaller numbers of timeouts, compared to the uplink experiments, because of the greater robustness of ACK packets to channel errors compared with data packets. In other words, since such robustness often allows a sender to successfully receive three duplicate ACKs while data packets are lost due to WiFi channel errors, the sender can invoke the fast recovery algorithm instead of waiting for the expiration of an RTO. However, when comparing MPTCP-MA with MPTCP, we observe that MPTCP-MA reduces the average number of timeouts by around 72%.

Even with the smaller numbers of timeouts, both types of MPTCP have a larger average peak RTO value in the downlink experiments than in the uplink experiments. This is because under both types of MPTCP, the sender often continues to transmit packets through an inactive subflow due to ACK robustness. Such transmitted packets result in significantly larger estimated RTTs, which affect the RTO values. In particular, MPTCP-MA using WiFi and 3G exhibits a much higher average peak RTO than MPTCP-MA using WiFi and LTE. That is, the larger RTT of a 3G connection can cause a more delayed suspension of a WiFi subflow than with LTE, resulting in a larger RTO value. However, MPTCP-MA still yields a smaller average peak RTO value than MPTCP.

Figure 2.16 presents the cumulative distribution function of RTO values for the WiFi subflow. When coupled with LTE, the RTO values of MPTCP-MA range up



(a) WiFi and LTE



(b) WiFi and 3G

Figure 2.16. CDF of RTO values for WiFi subflow (Downlink)

to five seconds, whereas those of MPTCP go up to a considerably larger 30 seconds. This is consistent with the uplink results. However, MPTCP-MA using WiFi coupled with 3G also yields large RTO values ($\geq$10 seconds) due to its notification delay: the receiver sends MP_SST to notify path status. We can expect that MPTCP will lose a fraction of the active period of the WiFi subflow while waiting for a successful transmission after such a large RTO expires. Recall that MPTCP-MA releases the backup mode of a subflow immediately after the subflow is labeled active and then starts transmitting packets without waiting for an RTO expiration. Therefore, these large RTO values cannot affect the performance of MPTCP-MA unless packets transmitted after releasing the backup mode are lost. However, in Figure 2.14, we already see that MPTCP-MA successfully obtains the gain from using the WiFi subflow in spite of this large RTO values by releasing the backup mode of the subflow when the subflow really is active, and thus avoids timeouts.

2.5 Related Work

Although MPTCP is being standardized by the IETF, there is no previous work considering an efficient MPTCP subflow management based on MAC-Layer information.

Goff et al. [23] propose Freeze-TCP to enhance TCP throughput in the presence of frequent disconnections and reconnections. Freeze-TCP assumes that a receiver can predict a temporary disconnection: the authors' example continuously monitors the received signal strength at the receiver-side. Once a disconnection is predicted, the receiver sends an ACK with an advertised window size of zero in order for the sender to freeze the TCP connection and start probing. However, the authors do not clarify how to predict such a temporal disconnection. Also, with a poor wireless link, a single-path TCP connection may have difficulty delivering control packets back to the sender.

Klemm et al. [38] propose mechanisms based on signal strength measurements to improve TCP performance in mobile ad-hoc networks. To alleviate packet losses due to mobility, their approach temporarily applies higher transmission power if the signal strength measurement indicates that a node is likely to move out of communication range.

Li et al. [40] suggest Link Signal Strength Agent Protocol (LSSA) to report measured signal strength to the TCP layer in mobile ad-hoc networks. However, it requires additional control packet exchanges which can fail when a single-path TCP connection is unreliable.

Shin et al. [70] propose a loss recovery scheme to differentiate between congestion loss and wireless loss using the MAC Management Information Base (MIB). In their scheme, a particular TCP packet is regarded as lost by wireless error if the number of frame transmission failures increases right after the packet is transmitted. However, it is difficult to identify which TCP connection is associated with a specific packet loss, since the MIB counters are across all packets and do not distinguish losses by connection.

Passach et al. [52] study the impact of mobile/WiFi handover performance with MPTCP. The authors investigated using different modes such as Full-MPTCP mode (where all potential subflows are used) and Backup mode (where only a subset of subflows are used to transmit packets). However, they do not explore how to quickly utilize all available paths and manage them when path connectivity frequently changes.

Raiciu et al. [62] also study mobility with MPTCP. They examined a mobile MPTCP architecture consisting of a mobile host, an optional MPTCP proxy, and a remote host. While they show that MPTCP outperforms standard TCP in a mobile scenario, they do not examine the under-utilization problem of recovered subflows.

2.6 Conclusions

In this chapter, we presented a method to improve MPTCP performance in the presence of intermittent path connectivity. We proposed an MPTCP variant, called MPTCP-MA, which manages subflows based on the status estimation using MAC-Layer information. After implementing MPTCP-MA in Linux on a desktop PC and on a mobile device, we performed experiments in an actual mobile scenario. Our experimental results show that MPTCP-MA is able to more quickly use active WiFi subflows than MPTCP, and thus can achieve significantly better performance.

CHAPTER 3

MODELLING MPTCP ENERGY CONSUMPTION BEHAVIOR FOR MOBILE DEVICES

Applying MPTCP to mobile devices introduces a new concern, namely, the additional energy consumption for operating multiple network interfaces. Mobile devices are frequently constrained by the amount of power available in their batteries. Processing power continues to grow exponentially, but battery storage increases slowly by comparison. Thus, power consumption management is an important area of research, particularly in mobile devices such as smartphones. In order to utilize MPTCP on mobile devices with limited energy resources, it is important to understand the power consumption behavior of MPTCP.

In this chapter, we examine MPTCP energy consumption behavior via a combination of measurement, modeling, and experimentation. We measure power consumption across a range of scenarios, varying download/upload size and available path bandwidth. We develop an energy model for MPTCP power consumption based on experimental measurements taken using MPTCP on a mobile device based on these measurements.

3.1 Background

3.1.1 Energy in Mobile and Wireless

Due to limited battery life of mobile devices, understanding and reducing energy consumption in mobiles has been an active area of research. Researchers have observed high power overheads in cellular networking [3, 30], which contributes to cellular interfaces being less energy-efficient than WiFi interfaces [30]. Other studies have discovered that more energy is consumed when the signal is weak [17, 69], and that multiple access points can cause extra power loss [47]. Researchers have examined WiFi in particular, reducing power consumption via rate adaptation [41], sleeping short periods [34], using fewer antennas [25], and lowering the clock rate [44]. A recent body of work [45, 56, 55, 60, 74] contributes tools and methodologies to measure power consumption on smartphones so as to identify sources of energy drain.

3.1.2 Cellular Promotion and Tail

Multiple works have identified and addressed the power overheads in cellular interfaces, often referred to as the *promotion* and the *tail* [3, 30]. The 3rd Generation Partnership Project (3GPP) standard defines a state machine for cellular interfaces, and describes the possible power states of each device connected to the network. Idle interfaces typically remain in a low-power state to save energy. Before a packet can be sent or received, an idle interface must switch from a low power state to a high power one, which takes additional time; this is referred to as the *promotion*. After a transmission is complete, however, a cellular interface does not immediately return to the lower power state. Instead, ostensibly to save energy, the interface remains in a high power state for a period of time. If there is no further packet transmission during that period, the interface then returns to the low power state. This period is referred to as the *tail*, and can last 6–12 seconds depending on the provider. Given its length, the tail contributes a significant portion of the energy cost for transmission, especially for short transfers. We refer to these as fixed energy overheads. Our mobile devices incur these overheads, as shown in Figure 3.1. Device specifications are given in Table 3.1.

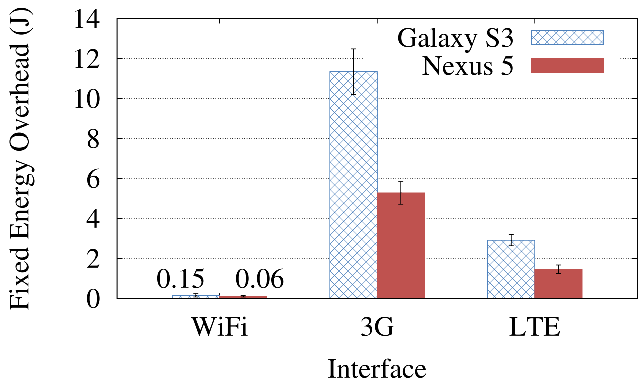


Figure 3.1. Fixed Energy Cost: WiFi and Cellular

Name	Samsung Galaxy S3	LG Nexus 5
Release Date	May 2012	Nov 2013
App. Processor	Qualcomm MSM8960	Qualcomm 8974-AA
Semiconductor	28nm LP	28nm HPM
Android Version	4.1.2 (Jelly Bean)	4.4.4 (KitKat)
Kernel Version	3.0.48	3.4.0
WiFi chipset	Broadcom BCM4334	Broadcom BCM4339

Table 3.1. Mobile Devices

3.2 Measurement Setup

We explore energy models for two mobile devices, the Samsung Galaxy S3 and the LG Nexus 5, listed in Table 3.1. All our measurements are based on an MPTCP implementation that was ported from Linux to Android [15, 42].

To measure the energy consumption of the mobile devices, we collect the voltage (V_S) and current (I_S) supplied to each device. Our measurements depend on the level of support available on the phone. The Nexus 5, a more recent phone, provides on-board chip voltage and current measurements which we can use directly. The Galaxy S3, which is 18 months older, does not provide such support. For the Galaxy S3, we built an electric circuit with an externally mounted battery and a high precision resistor ($R = 0.05\Omega$) between the battery and the mobile device. We use a National Instruments Data Acquisition system (DAQ), with a sampling rate of 100K samples/s, in order to measure V_S and the voltage drop across the resistor, V_R; I_S then is given by $I_S = V_R/R$.

We collect energy consumption of uploads and downloads of files with sizes varying from 16 KB to 16 MB, with ten iterations per transfer. In these experiments, we force the mobile device CPU to remain in performance mode so as to avoid energy consumption changes due to the CPU switching power modes. We also set the display brightness of the mobile device to a fixed level. We compute energy consumption solely for packet transfer by subtracting baseline energy consumption (without packet transfer) from the measurements.

3.3 Single-Path Energy Model

We use a standard regression-based energy model [3, 30] to model single-interface energy consumption during data transfer. Energy consumption per transferred byte is represented as a function of the observed TCP throughput, with different parameters for uploads and downloads. We denote this regression model as the base regression model. Given a download throughput over an interface of B (Mbps), the fitted value of the energy consumption for downloading each byte, P ($\mu J/B$), is

$$P(B) \;=\; \alpha \times B^{\beta},$$

where α and β are device dependent. We include the consumed energy for sending or receiving an ACK as part of the overall energy consumption for receiving or sending data packets.

Table 3.2 shows the α and β values for our two devices, for both uploads and downloads, using WiFi, HSPDA (3G), and LTE (4G), on the AT&T cellular network. Note that we place the devices so that they observe good signal strength, $\geq$ -95 dBm for 3G and $\geq$ -105 dBm for LTE, and $\geq$ -45 dBm for WiFi with no contention. We

		HSDPA	LTE	WiFi
Galaxy S3	α^d	9.344	10.043	4.675
Download	β^d	-0.929	-0.891	-0.818
Galaxy S3	α^u	12.529	13.344	3.614
Upload	β^u	-0.852	-0.836	-0.662
Nexus 5	α^d	5.398	4.679	1.789
Download	β^d	-0.554	-0.851	-0.542
Nexus 5	α^u	11.454	7.683	2.642
Upload	β^u	-0.375	-0.712	-0.646

Table 3.2. Single-Path Regression Coefficients

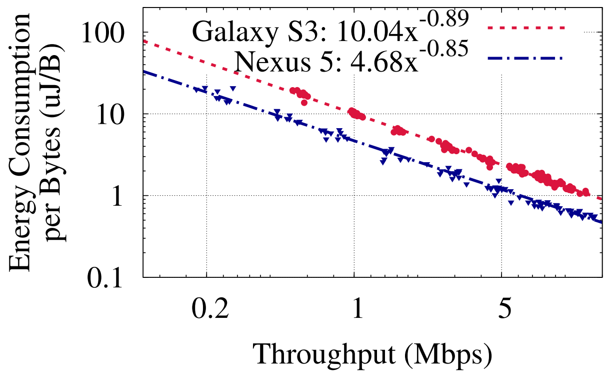


Figure 3.2. Single-Path TCP Per-Byte Energy Consumption: LTE on AT&T

observe that the regression model $P = \alpha \times B^\beta$ yields estimates of P close to the actual measured values as a function of B, the available throughput. An example showing the fit is given in Figure 3.2, for downloads over LTE.

We observe that the Nexus 5 exhibits notably lower per byte energy consumption for both 3G and LTE than does the Galaxy S3, presumably because of the newer technology as described earlier. In contrast, the per-byte energy consumption of WiFi does not significantly differ between the devices. This is because, as shown in Table 3.1, the 3G/LTE radio is integrated in the devices' system-on-chips manufactured using different semiconductor technologies. The devices' WiFi uses similar chipsets, which may explain why they exhibit similar energy consumption behavior.

Since recent phones supply detailed power information, we do not believe generating this model is a significant problem. Researchers have even proposed automated methods for deriving this information [77, 80], which unfortunately were not available to us when we conducted this study. Recent versions of Android also contain a *power_profile.xml* file provided by the manufacturer that specifies this information explicitly.

3.3.1 Effect of WiFi Channel Condition

Network interfaces consume larger amounts of energy to transfer packets as signal strength decreases, due to MAC or TCP layer retransmissions [17, 69, 73]. A network interface also dynamically changes its physical layer bit rate or transmit power according to the channel status, which can yield different energy consumption behavior. In the following subsections, we examine energy consumption under different channel conditions, such as different signal strength and contention levels.

First, we investigate energy consumption per transferred byte as a function of throughput controlled by different WiFi signal strength and contention level. We vary WiFi signal strength by locating the mobile device (the Galaxy S3) at different distances from the AP and we introduce different level of contention by using $N = 0, ..., 3$ nodes generating UDP traffic, which contend for the same WiFi channel as the mobile device. Figure 3.3 presents the measured energy consumption per transferred byte as a function of obtained TCP throughput according to different signal strengths and numbers of contending nodes: smaller signal strength and more contending nodes results in lower throughput. Note that Figure 3.3 shows the results of selected throughput range ([4, 8] Mbps for downloads and [2, 4] Mbps for uploads). The solid red line is the regression result that we obtained for the Galaxy S3 in the previous section. The dashed lines are the corresponding results with throughputs controlled by signal strength or contention level.

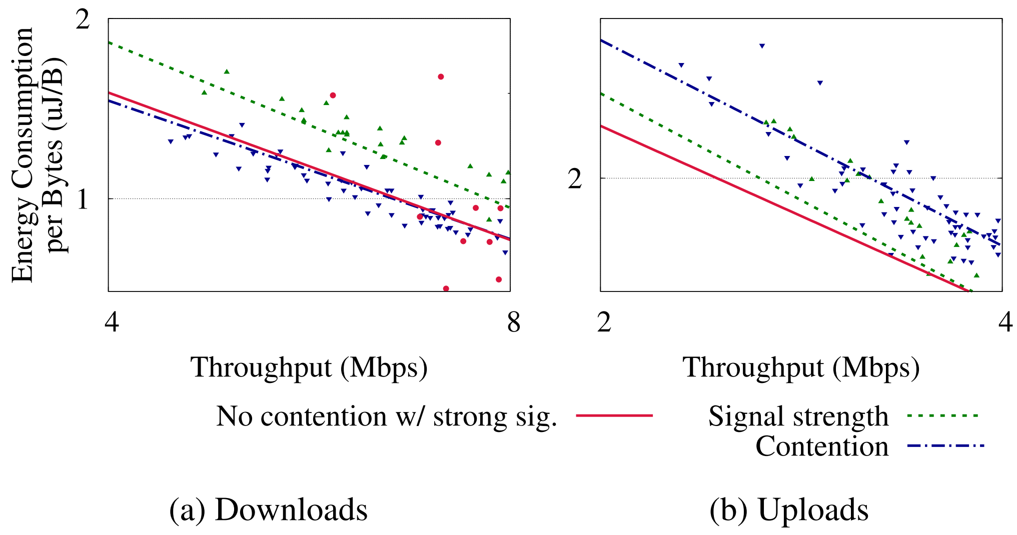
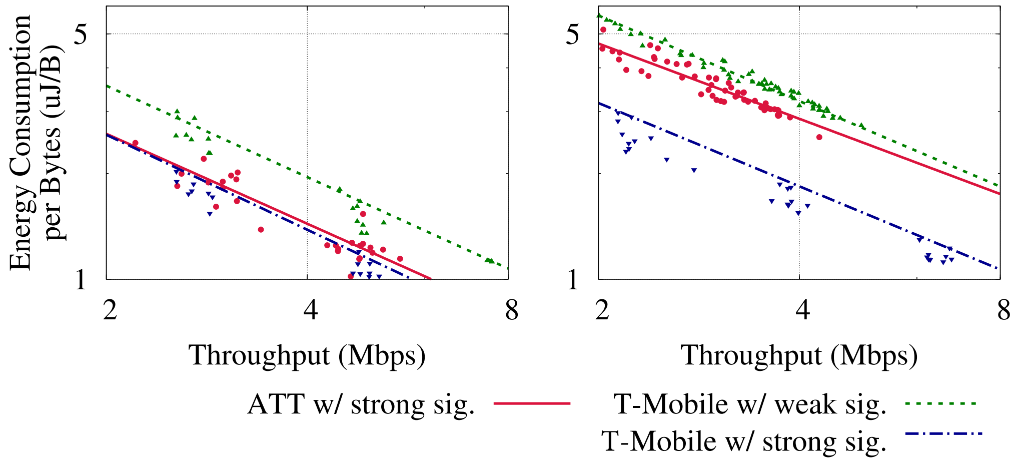


Figure 3.3. Effect of Signal Strength and Contention on Energy Consumption for WiFi Packet Transfer (Log-Log scale, Galaxy S3)

In the case of downloads, we observe that the device consumes slightly more energy per transferred byte with weaker signal strength (see the red solid and green dashed line in Figure 3.3(a)) whereas the level of contention does not affect per-byte energy consumption (compare the red solid and blue dashed line in Figure 3.3(a)). We conjecture that this is because the device reactively consumes energy for receiving packets. In other words, when signal strength is low, the 802.11 MAC layer at the AP tries to retransmit frames until successfully delivered, i.e., the device continues to spend energy for receiving frames even though it cannot decodes them due to weak signal strength. In addition, since weak signal strength can result in ACK delivery failures from the device to the AP, the device can consume energy to receive unnecessary frames that it has already acknowledged. In contrast, in the presence of large contention in the WiFi channel, the AP defers frame transmissions until it finds an empty time slot using the 802.11 backoff algorithm [66]. In other words, the device does not consume energy for receiving while the AP is doing the backoff procedure.



(a) Downloads (b) Uploads

Figure 3.4. Effect of Signal Strength and Carrier on Energy Consumption for LTE Packet Transfer (Log-Log scale, Nexus 5)

In the case of uploads, both weak signal strength and large channel contention increases energy consumption per transferred byte. This may be because the device proactively operates its interface as a sender; as opposed to the download case, the device continues to consume energy in order to retransmit frames or scan the WiFi channel.

3.3.2 Effect of Cellular Channel Conditions

We also investigate the effect of cellular (LTE) channel conditions on energy consumption per transferred byte. Since we cannot control signal strength and contention levels in the LTE channel, we conduct experiments at two locations with different LTE signal strengths. We also examine the effect of the choice of cellular carriers on per-byte energy consumption. To this end, we locate the mobile device, a Nexus 5, which supports both AT&T and T-Mobile LTE, at two locations: LTE signal strength is stronger at one place than at the other ($\geq$-105 dBm vs. $\geq$-112 dBm), denoted as strong and weak signal strength.

Figure 3.4 presents the measured energy consumption per transferred byte as a function of TCP throughputs (in the selected range of [2, 8] Mbps) under the different LTE signal strengths and carriers. Different TCP throughputs are obtained by using different file sizes. As shown in Figure 3.4, signal strength is a more important factor than the carrier, and its effect is more significant for uploads than for downloads (compare the green and blue dashed line in Figure 3.4 (a) and (b)). When signal strength is high, both carriers yield almost the same energy consumption behavior for downloads (see the red solid and blue dashed line in Figure 3.4(a)), while for uploads T-Mobile exhibit lower energy consumption than AT&T. Note that the strong signal strength is not defined exactly the same for each carrier, since we do not have fine-grained control on the LTE signal strength: the strong signal strength for T-Mobile (around -98 dBm) is higher than that for AT&T (around -105 dBm). Comparing different signal strengths of T-Mobile, we observe that the weak LTE signal strength yields slightly higher energy consumption per transferred byte for downloads whereas it notably increases per-byte energy consumption for uploads. This is because LTE networks have uplink transmit power control according to channel status such as path loss and signal fading: the transmit power at the device is set based on the signal strength or feedback from cell towers. Thus, the device is likely to increase its uplink transmit power as the LTE signal strength from the cell towers becomes weaker [71]. Since the device only sends TCP ACK packets when downloading a file, energy consumption per transferred byte does not increase significantly. However, in the case of uploads, such a larger transmit power results in significant increase of per-byte energy consumption since the device transmits data packets, which are larger than ACK packets.

3.4 Multi-Path Energy Model

A simple estimate of MPTCP energy consumption would be to sum the energy consumed by each interface over the transferred data. However, we observe that the actual measured MPTCP energy consumption is less than this estimate in devices such as the Galaxy S3, possibly due to the energy consumption for shared components while operating multiple network interfaces such as CPU. To account for such shared energy consumption, we assume that a device consumes a fraction γ of the sum during the overlapped period for packet transfer. In the case of the fixed energy overhead for promotion and tail, we assume that the overhead is separately consumed for each interface.

Let B_C and B_W denote the throughputs of 3G/LTE and WiFi, respectively. Suppose that S is the size of the transfer and S_W and S_C are the number of bytes transferred through WiFi and 3G/LTE, respectively; $S = S_W + S_C$. Let θ denote the proportion of the duration of the transfer when both interfaces are simultaneously active. Given S_W and S_C, we approximate θ as

$$\theta = \frac{\min\left(S_W/B_W, S_C/B_C\right)}{\max\left(S_W/B_W, S_C/B_C\right)}.$$

Based on this approximation, we estimate the MPTCP energy consumption during packet transfer, E_T, as

$$E_T \quad = \quad \left(P_W(B_W) \times S_W + P_C(B_C) \times S_C\right)(1 - \theta + \gamma\theta),$$

where B_W and B_C are the available throughputs over WiFi and 3G/LTE and S_W and S_C are the transferred bytes over WiFi and 3G/LTE, respectively. The total MPTCP energy consumption, including the energy overheads associated with the promotion and tail states of each interface, is then represented as

$$E_M \quad = \quad E_T + C_W + C_C,$$

where C_W and C_C are the fixed energy overheads for the promotion and tail states of the WiFi and 3G/LTE interfaces, respectively.

3.4.1 Determining γ

To determine γ, the fraction of shared energy consumption, we perform a set of experiments to measure MPTCP energy consumption while a device downloads or uploads files of various sizes using MPTCP with WiFi and LTE. We choose γ to minimize the mean square error between measured and estimated[1] values. Figure 3.5 shows the mean square error of energy consumption as a function of γ when the Galaxy S3 and the Nexus 4 use MPTCP with WiFi and 3G(Nexus 4)/LTE(Galaxy S3). In the case of the Galaxy S3, the mean square error is minimized when γ is equal to 0.8586. In other words, approximately 15% of energy appears to be consumed by shared components when MPTCP is simultaneously operating over both interfaces. One example of shared energy consumption might be CPU cycles used to process MPTCP instructions. The S3 uses a Qualcomm Snapdragon processor [67, 75] with 3G/LTE integrated on the chip, thus, it seems likely that they share some energy while operating.

The Nexus 5, on the other hand, has a γ of one, namely there is no shared portion of energy consumption. The two devices use different chipsets and generations of technology, thus γ is specific to a device.

3.5 Model Validation

To validate our MPTCP energy model, we compare measured MPTCP energy consumptions with model estimates. We use the Galaxy S3 and Nexus 5 in this validation, focusing on the case where MPTCP simultaneously uses the WiFi and LTE interfaces.

[1]We use the base regression results obtained in §3.3

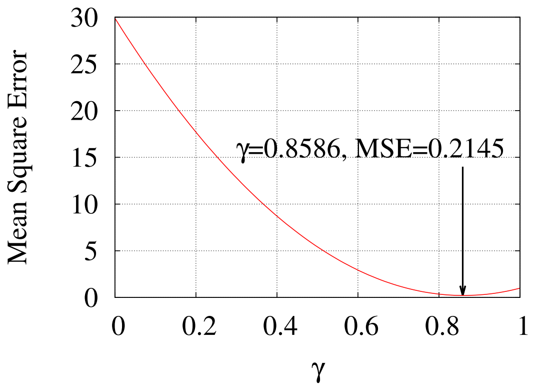


Figure 3.5. γ that minimizes mean square error ($\gamma = 1$ in case of Nexus 5)

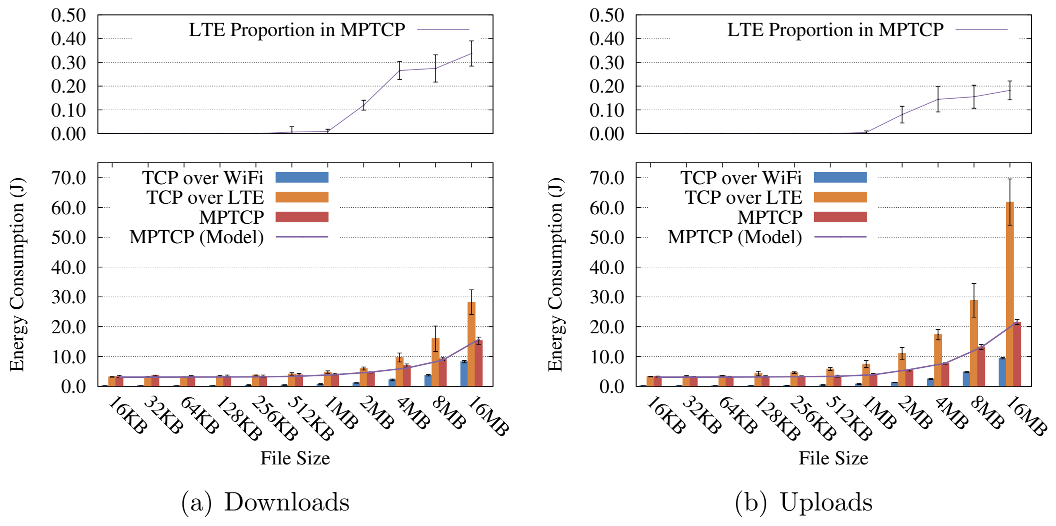


Figure 3.6. Total Energy Consumption according to File Size (Galaxy S3)

Figure 3.6 shows the total download energy consumption vs. file size (averaged over ten iterations) for three configurations: TCP over WiFi alone, TCP over LTE alone, and MPTCP utilizing both interfaces (We only include download results with selected file sizes using Galaxy S3 for AT&T; other experiments show similar behavior). Figure 3.6 also shows the energy consumption predicted by our model. We observe that our model accurately estimates the energy consumption of MPTCP: the root mean square errors normalized by the average measured energy consumption are less than 17% as shown in Figure 3.7.

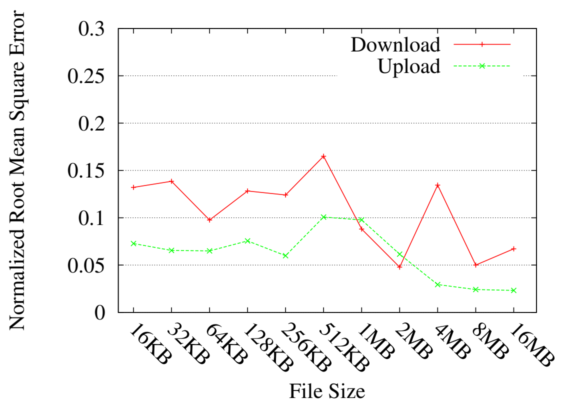


Figure 3.7. Normalized RMSE

We also observe that MPTCP is less energy efficient than single-path TCP over WiFi as the file size decreases. This is because of the fixed costs associated with the promotion and tail state of LTE: when the file size is small, the device spends energy to establish an LTE subflow, even though it rarely transmits packets over LTE. MPTCP does not even utilize an LTE subflow until the transfer size becomes larger than 512 KB. This is due to the relatively large RTT (about 65 ms for downloads and 95 ms for uploads) over the LTE path; setting up a subflow over LTE takes longer than over WiFi, which has much lower RTT (about 15 ms). The larger RTT also means it takes longer for the congestion window of the LTE subflow to increase in size [7]. Thus, a small file ($<$ 512 KB) transmission completes in a short time and consumes a relatively small amount of energy for packet transfer compared to the fixed overhead. Consequently, MPTCP yields similar energy consumption to TCP over LTE when a file is smaller than 512 KB, which is significantly larger than that of TCP over WiFi, even though it allocates almost of all the traffic to the WiFi interface. We see that in the case of small file transmissions, MPTCP wastes power, whereas TCP over WiFi is much more efficient.

3.6 Related Work

Balasubramanian et al. [3] measure energy consumption on a Nokia N95 platform and identify high energy overhead as the result of the tail state in 3GPP interfaces (GSM, 3G). They develop a protocol called *tailender* to schedule transfers so as to minimize energy consumed by the tail.

Halperin et al. [25] examine the relationships between power consumption and channel width, number of attennas, and spatial streams in 802.11n.

Caroll and Heiser [5] provide an analysis of power consumption in smartphones, measuring component power costs across a variety of workloads. They perform a more recent study [6] for the same smartphone that we use, the Samsung Galaxy S3.

Schulman et al. [69] show that the power consumed by wireless radios is higher when the signal is week. They present Bartendr, a system for scheduling transmissions when the signal is strong so as to minimize power consumption. Ding et al. [17] provide a more in-depth analysis of the impact of signal strength.

McCullough et al. [48] examine the accuracy of several linear-regression based power models for a Nehalem-based server. They present some improvements in modeling and show how multiple cores complicate energy modelling.

Garcia-Saavedra et al. [22] show that a substantial fraction of energy consumed on 802.11 devices is proportional to the transitions across the I/O bus, rather than per-packet or per-byte. This suggests optimizations (e.g., TCP segmentation offload) that amortize these costs may have power savings in addition to CPU savings.

Huang et al. [30] provide an in-depth look at power and performance characteristics of 4G LTE networks based on a large-scale measurement of a commercial cellular provider. They show that while LTE is more energy-efficient than 3G, it is still not as efficient as WiFi, partially due to the tail state cost in LTE.

Raiciu et al. [62] look at a number of issues in using MPTCP for mobility, including power consumption. They propose and evaluate a simple strategy that periodically samples both paths for 10 seconds and then uses the more energy-efficient path for 100 seconds. Evaluating their strategy via simulation, the approach is more power efficient than an energy-unaware MPTCP implementation, but achieves lower bandwidth.

Paasch et al. [52] study mobile/WiFi handover performance with MPTCP. They also measure energy consumption on a Nokia N950 smartphone for two download scenarios and find that using WiFi alone is more energy-efficient than base MPTCP.

3.7 Conclusion

In this chapter, we presented a through analysis on energy consumption behavior of MPTCP. We conducted energy consumption measurements varying available bandwidth, wireless channel condition such as signal strength and contention. Based on these measurements, we built a regression model for MPTCP energy consumption behavior. Our experimental results show that the model accurately estimates the energy consumption of MPTCP with normalized root mean square errors less than 17%

CHAPTER 4

IMPROVING ENERGY EFFICIENCY OF MPTCP FOR MOBILE DEVICES

In many environments, MPTCP does not reduce power consumption compared to single-path TCP. This is because MPTCP does not consider the per-byte energy efficiency of a network path, which depends on several factors, including available bandwidth and the interface type. It also ignores the high fixed energy costs of activating cellular interfaces, known as the promotion and tail costs [3].

In this chapter, we shed light on the energy behavior of MPTCP on smartphones. We seek to address the following questions: 1) Are there environments or scenarios where MPTCP is more energy efficient than single-path TCP? 2) If so, how can we recognize them and take advantage of them? Can we improve MPTCP's energy efficiency?

Based on the MPTCP model in Chapter 3, we determine conditions under which MPTCP is more energy-efficient than either standard TCP or MPTCP. Informed by this finding, we design and implement an energy-aware eMPTCP which we have implemented on an Android platform, the Samsung Galaxy S3. Previous approaches in this area have either been based on simulation [58] or have looked at much more restricted operating environments [4, 52, 62]. eMPTCP operates on implemented on the phone and requires no offloaded computation at run time in contrast to [58]. In addition, none of the above attempt to reduce cellular fixed overheads. Ours is the first broad evaluation of an implemented, deployable energy-aware MPTCP.

4.1 Energy Aware MPTCP

4.1.1 Overview

eMPTCP's goal is to reduce energy consumption compared to MPTCP, while minimizing additional delay. It does so by dynamically choosing paths based on estimated energy efficiency. It also tries to avoid unnecessary cellular fixed costs by delaying subflow establishment over cellular interfaces, unless it believes it is more energy efficient to do so.

To this end, eMPTCP requires an energy model that captures power consumption according to network interface usage. There is a large body of literature that considers mobile devices' energy consumption based on bandwidth, signal strength, simultaneous use of interfaces, and so on [3, 17, 30, 42, 49]. We utilize our parameterized energy model described in Chapter 3 to generate the required information for eMPTCP.

Figure 4.1 presents the architecture of the eMPTCP implementation. Four core components are added to the regular MPTCP at the transport layer: The *bandwidth predictor* (§4.1.2) monitors bandwidths over all active subflows and predicts future bandwidths. The *energy information base* (§4.1.3) holds the transition thresholds, indexed by bandwidth, for selecting which interfaces to use. The *path usage controller* (§4.1.4) makes decisions about which subflows to utilize, based on information retrieved from the bandwidth predictor and the energy information base. The *delayed subflow* (§4.1.5) module manages subflow establishment requests and delays them when necessary, based on information from the other two components. We describe each component in turn.

4.1.2 Bandwidth Predictor

The bandwidth predictor samples all active subflow throughputs and predicts their future values. By querying the routing information at the Internet layer, the

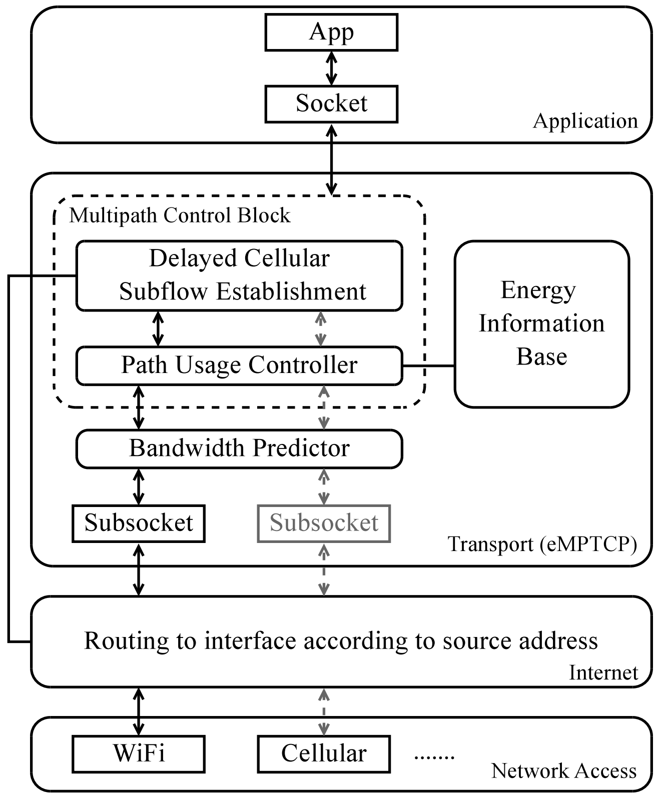


Figure 4.1. eMPTCP Architecture

bandwidth predictor identifies the interfaces associated with active subflows and categorizes predictions per interface. Each subflow is sampled periodically with a sampling interval δ, which is based on the subflow's measured round-trip time (RTT) during subflow establishment, i.e., the elapsed time for the TCP three way handshake. Throughput predictions are made using a Holt-Winters time-series forecasting algorithm [65], which is known to be more accurate than formula-based predictors [28].

If an interface is inactive, the bandwidth predictor will not have any current throughput information about that interface. This can occur for two reasons: because the interface was activated but is now deactivated, or because it has never been activated. If the interface was deactivated, the bandwidth predictor uses old observed samples together with new sampled throughputs. If the interface has *never* been

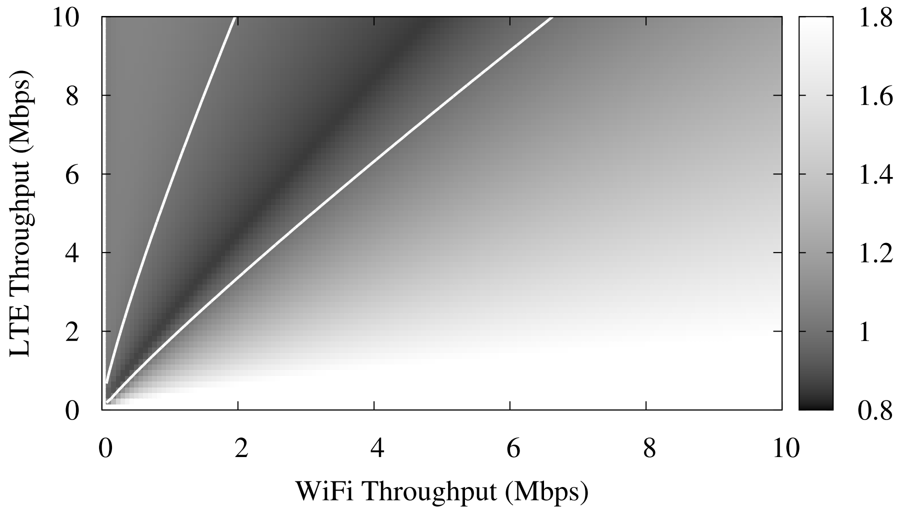


Figure 4.2. Energy Efficiency per Downloaded Byte (Samsung Galaxy S3)

activated, the predictor assumes non-zero throughput (e.g., 5 Mbps) as an initial bandwidth for the interface to allow eMPTCP to probe the path through the interface.

4.1.3 Energy Information Base

The Energy Information Base (EIB) contains the data required for eMPTCP to decide which interface(s) to use to maximize energy efficiency. Since we cannot predict the amount of data remaining to be transferred, eMPTCP assumes a large transfer and defines efficiency in terms of per-byte energy consumption. This information is computed offline using our parameterized energy consumption model that accounts for multiple interfaces [42] and generates the EIBs for two mobile devices listed in Table 3.1. Note that any energy model generated by state-of-art techniques such as [77, 80] can be used to populate the EIB.

Given the energy model, we are able to characterize the operating region where MPTCP is more energy efficient than standard single-path TCP over Cellular or WiFi. For example, Figure 4.2 shows the *relative* energy consumption per downloaded byte

LTE Thpt. (Mbps)	WiFi throughput (Mbps) LTE Only Threshold	WiFi Only Threshold
0.5	< 0.043	≥ 0.234
1.0	< 0.134	≥ 0.502
1.5	< 0.209	≥ 0.803
2.0	< 0.304	≥ 1.070
...	......	

Table 4.1. Example of Energy Information Base

over both WiFi and LTE on the Samsung Galaxy S3, normalized by the amount of energy consumed using the best single interface (WiFi or LTE). The figure is a grey-scale heat map where the darker the area, the more energy-efficient MPTCP is. At the left side of the region, TCP over LTE is the most energy efficient, whereas on the right side TCP over WiFi is the most efficient. In the region defined by the "V" shape, inside the solid white curves, using both interfaces consumes the smallest amount of energy to download a byte. This heat map is instantiated in the EIB.

The EIB represents this data as an array, indexed by the observed LTE throughput, where each entry includes two WiFi throughputs. This pair of throughputs specifies the transition points where eMPTCP should switch from a single interface to multiple interfaces and vice versa. For example, in the second row in Table 4.1, given an observed LTE throughput of 1 Mbps, if the observed WiFi throughput $<$ 0.134 Mbps, LTE-only should be used; if the observed WiFi throughput $\geq$ 0.502 Mbps, WiFi-only should be used; otherwise, both should be used.

4.1.4 Path Usage Controller

The path usage controller dynamically decides which paths to use based on energy efficiency. Once a multi-path connection is established, eMPTCP dynamically decides which interfaces to use by retrieving the current estimates from the bandwidth predictor and using them to query the EIB. If data needs to be queued on a subflow, the subflow with the highest per-byte energy efficiency is chosen. Thus, eMPTCP seeks a local optimum in terms of total energy consumption.

To prevent oscillations, our algorithm uses a 10% ‘safety factor’ when switching from one state to another. Continuing the earlier example (second row of Table 2), if eMPTCP is using both interfaces, it requires a predicted WiFi throughput of 0.552 Mbps, not 0.502, to transition to WiFi-only. Similarly, if eMPTCP is using WiFi-only, it requires a predicted WiFi throughput of 0.452 Mbps to switch to both interfaces. This adds some hysteresis to the system and prevents it from switching states too frequently. Note that, eMPTCP does not typically switch to using a cellular interface only, since the expected gain is not much more than using both.

4.1.5 Delayed Subflow Establishment

When data transfers are small, we wish to avoid any unnecessary expenditure of energy establishing a cellular subflow, with the attendant energy costs of the promotion and tail states. To this end, if a cellular interface is not already active, eMPTCP introduces a delay between WiFi and 3G/LTE subflow establishment. The cellular subflow is not started until after eMPTCP receives κ bytes through the WiFi interface, thus avoiding these fixed overheads when transferring data fewer than κ bytes.

Even after κ bytes of data have been transferred, eMPTCP still postpones establishing cellular subflows if the measured WiFi throughput is large enough such that using WiFi-only is more energy-efficient than using both interfaces, as indicated by the energy information base.

If the available WiFi throughput is low, solely using the WiFi subflow until κ bytes are transferred can be less energy efficient than using both interfaces. Indeed, κ may never be reached on a slow WiFi subflow. To preclude this scenario, eMPTCP also uses a timer to trigger cellular subflow establishment. If the timer with value

τ expires, eMPTCP establishes a cellular subflow, even if fewer than κ bytes have been transferred. The time threshold τ needs to be tuned to correctly estimate WiFi throughput: τ must be larger than or equal to the time required to collect enough samples after the WiFi subflow throughput stabilizes. Suppose B_W is the available throughput over WiFi, W_{init} the initial congestion window size, R_W the WiFi RTT, and ϕ the number of required samples. Then the condition for τ is:

$$\tau \quad \geq \quad R_W \times \left(\log_2 \frac{B_W \times R_W + W_{init}}{W_{init}} + \phi \right) \tag{4.1}$$

eMPTCP also postpones cellular subflow establishment if the current MPTCP connection is in an idle state with no transmission activity, even if the timer expires. This avoids unnecessary cellular subflow establishment for idle connections, as some applications (e.g., HTTP [19]) hold connections open in idle states even after completing data transfer. eMPTCP regards a connection as idle if it does not send or receive any packets during an estimated RTT.

4.1.6 Implementation

We have implemented eMPTCP in the Android versions specified in Table 3.1. We focus on downloads over WiFi and LTE interfaces since they are the most common. Each component of eMPTCP utilizes routing information at the Internet layer to identify the interface associated with a subflow (`subsocket`). eMPTCP checks the destination entry of the socket. The destination entry (`struct dst_entry`) includes a pointer to the related network device (`struct net_device *dev`) which has a name of the interface and a pointer to the wireless device information (`struct wireless_dev *ieee80211_ptr`). Thus, eMPTCP can identify whether a socket is associated with a WiFi interface or not by checking if `dev` has a valid `ieee80211_ptr`.

MPTCP requires a default primary interface with which to initiate and receive transfers. Since WiFi is more energy efficient than cellular on our equipment and has negligible promotion and tail costs, as shown in Figure 3.1, we use WiFi as the default interface.

Once eMPTCP decides path usage, it accordingly adds an MP_PRIO option [21], which changes the priority of subflows, to the next packet to be transmitted; for example, MP_PRIO indicating that LTE subflow is in low priority is added in the next packet if using LTE is energy inefficient.

When the device re-uses a subflow over an interface suspended by the path usage controller, the sender needs to be able to start using this subflow quickly. To this end, eMPTCP disables the CWND reset after an idle period longer than the retransmission timeout in RFC2861 [27] to ensure that a subflow avoids unnecessary slow-start when eMPTCP starts re-using the subflow. In addition, eMPTCP sets the measured round trip time (RTT) of the new subflow to zero. This modification enables a renewed subflow to be quickly probed by the MPTCP scheduler, since it selects a subflow with the lowest RTT for packet transmission [63]. Note that this is only for *re-used* subflows; new subflows behave the same as standard MPTCP in terms of CWND and RTT.

4.2 Evaluation in a Controlled Lab

The goal of our first set of experiments is to investigate the behavior of eMPTCP in an environment where we can control wireless conditions. We first examine a static configuration (§4.2.2), then vary the WiFi bandwidth (§4.2.3). We examine the impact of background traffic (§4.2.4), and then the impact of mobility(§4.2.5). We explore how well eMPTCP adaptively controls path usage to obtain greater energy efficiency than standard MPTCP, as well as the impact on download time.

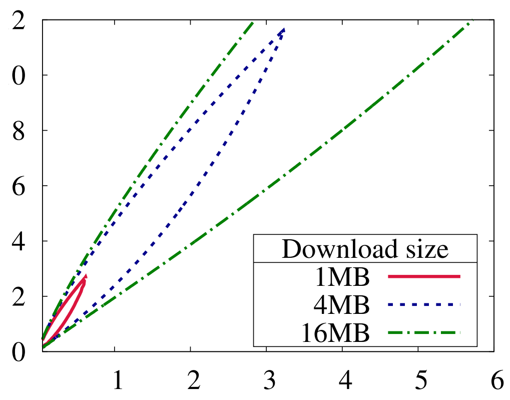


Figure 4.3. Operating Region where MPTCP is the Most Energy Efficient to Complete an Entire Transfer (Samsung Galaxy S3)

4.2.1 Experimental Setup

The mobile devices access a wired server, running Ubuntu Linux 12.04 with the MPTCP implementation [63]. The server is connected to our campus network through a single Gigabit Ethernet interface. Mobile devices can communicate with the server over the Internet using a WiFi access point (IEEE 802.11g), 3G or LTE cellular interfaces from AT&T. Energy traces are collected using the techniques described in Chapter 3.

For the controlled experiments with bandwidth changes, the mobile device downloads a large file while experiencing changing network conditions. We compare eMPTCP to standard MPTCP and TCP over WiFi in several scenarios, using the Samsung Galaxy S3.

We set eMPTCP parameters as follows. The download amount threshold κ to postpone an LTE subflow is set to one MB. This is because MPTCP is rarely more energy efficient than single path TCP when downloading a smaller sized file. Figure 4.3 presents the calculated throughput region where MPTCP is more energy efficient than TCP over WiFi-only and LTE-only, for 1 MB, 4 MB, and 16 MB downloads. Based on our energy model in Chapter 3, the regions inside the curves of each size correspond to the throughput values where MPTCP is more energy efficient than single-path TCP to download a file of that size. The timer threshold τ is set to three seconds: given our experimental setting, the estimated condition based on equation 4.1 to guarantee ten bandwidth samples is $\tau \geq 2.67s$. While these values have worked well for our experiments, refining them to improve performance remains a subject for future work.

4.2.2 Experiments with Static Configuration

The purpose of these experiments is to show that, in relatively simple static environments, eMPTCP makes the proper path decisions to reduce power consumption. We measure energy consumption and download time of eMPTCP, MPTCP, and TCP over WiFi for two extreme cases: persistent high (>10Mbps) and low (<1Mbps) WiFi bandwidth while the device downloads a 256 MB file at a fixed location. Figures 4.4 and 4.5 compare energy consumption and download times averaged over five runs for the two cases. In the first case, WiFi bandwidth is high, and thus using it is more power efficient than using either LTE or both interfaces. Figure 4.4 shows that eMPTCP chooses WiFi only, effectively behaving similar to single-path TCP over WiFi. In contrast, when WiFi bandwidth is small (<1Mbps), and thus less energy-efficient than LTE, Figure 4.5 shows that eMPTCP yields almost the same performance as MPTCP by using both interfaces (after the LTE startup delay determined by parameters κ and τ). This illustrates that eMPTCP automatically seeks the most energy efficient path usage without user involvement.

4.2.3 Experiments with Bandwidth Changes

Next we examine how robust eMPTCP is to changes in network bandwidth. Here, WiFi link bandwidth is modulated by a two state on-off process with exponentially distributed times spent in the on or off state with a mean of 40 seconds. The bandwidth provided by the AP is less than 1 Mbps or greater than 10 Mbps, depending

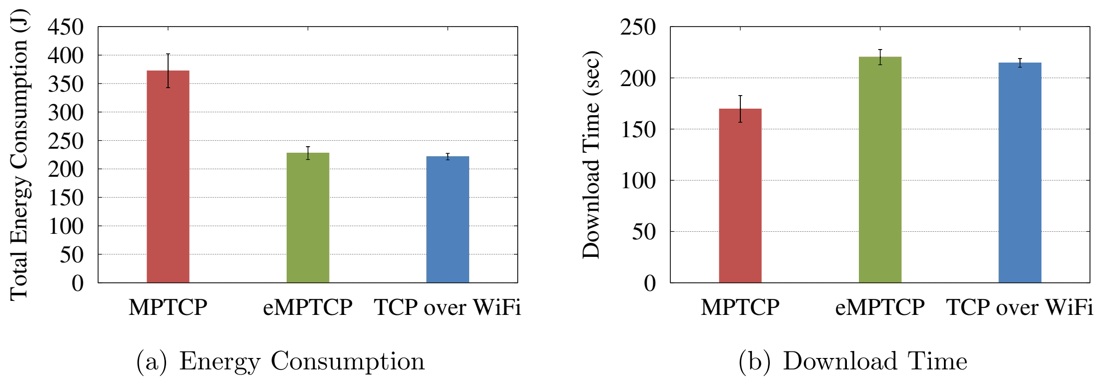


Figure 4.4. Static Good WiFi Comparison

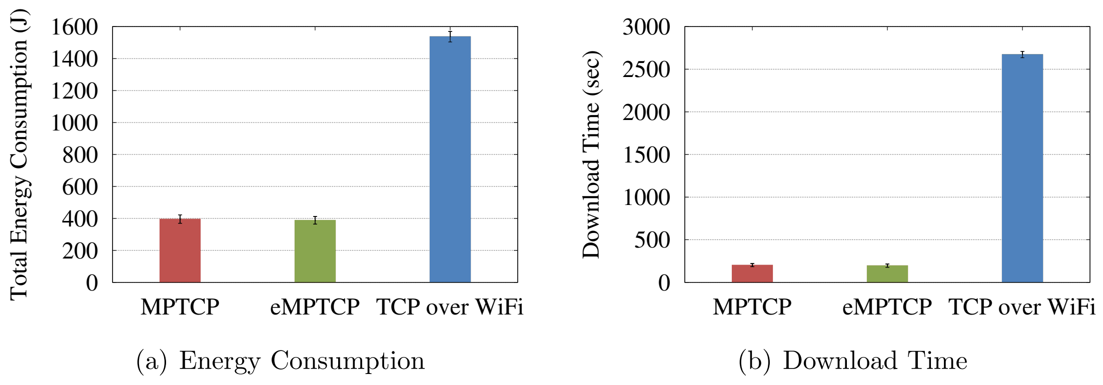


Figure 4.5. Static Bad WiFi Comparison

on its state. Measurements are taken while the device downloads a 256 MB file at a fixed location.

Figure 4.6 presents a time series trace of the accumulated energy consumption of eMPTCP, MPTCP, and TCP over WiFi for a single run with random WiFi bandwidth changes. At the beginning of the trace, after the LTE startup delay, eMPTCP uses both interfaces since WiFi throughput is too small to be energy efficient. However, eMPTCP suspends the LTE subflow after the WiFi bandwidth increases at around time 60, while MPTCP continues to use both interfaces. By suspending the LTE subflow, eMPTCP spends approximately 20% less energy than standard MPTCP at the cost of a 40% larger download time. Compared to single-path TCP over WiFi,

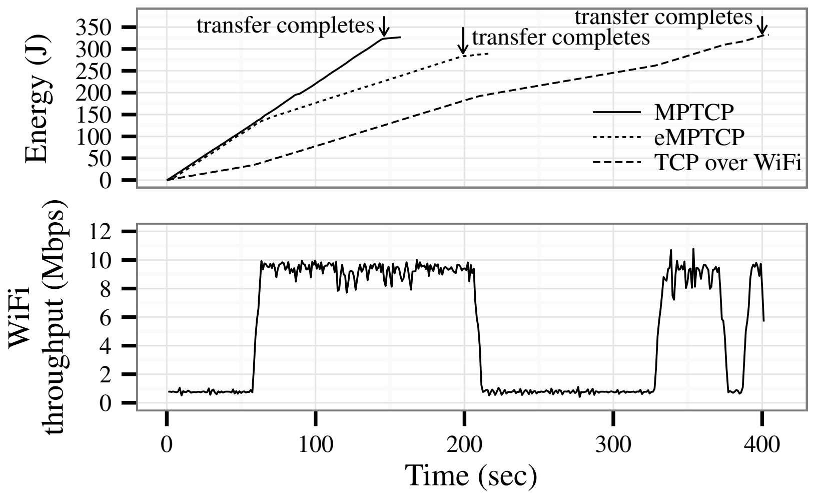


Figure 4.6. Accumulated Energy Consumption Example with Random WiFi Bandwidth Change

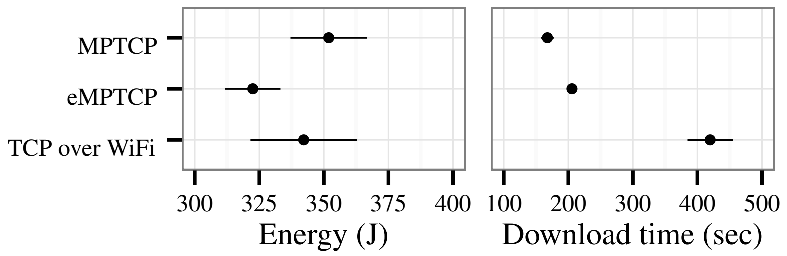


Figure 4.7. Random WiFi Bandwidth Changes

eMPTCP both completes the download sooner (by about 50%) and consumes less energy (by about 15%).

Figure 4.7 compares energy consumption and download time averaged over ten runs. This figure, as well as similar ones to follow, uses a symbol to show the sample mean together with horizontal bars of length twice the standard error of the mean (SEM). The quantity SEM for n samples $(x_1, x_2, \ldots, x_n)$ is given by the expression $s/\sqrt{n}$ where

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2} \tag{4.2}$$

is the sample standard deviation and $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the sample mean. In this set of experiments, eMPTCP consumes approximately 8% and 6% less energy on average than MPTCP and TCP over WiFi, respectively. However, eMPTCP is approximately 22% slower on average than MPTCP since eMPTCP utilizes the LTE subflow only when it can improve energy efficiency. In contrast to single-path TCP over WiFi, by utilizing an LTE subflow when WiFi throughput is less than 1 Mbps, eMPTCP completes downloads twice as fast and consumes less energy. Note that the performance gain achieved by eMPTCP differs according to how WiFi bandwidth changes. If WiFi bandwidth changes frequently, the switching overhead in eMPTCP may become noticeable as an LTE interface triggers a promotion and tail state each time that it starts to be used again.

4.2.4 Experiments with Background Traffic

In this section we investigate how well eMPTCP copes with random background traffic. It is well known that multiple WiFi nodes can contend for the air channel, causing interference and loss (e.g., [47]). Background traffic causes contention and interference in the communication channel, resulting in throughput changes similar to link bandwidth changes. In these experiments, we utilize two or three interfering sources, which use the same WiFi channel as the mobile device. While the device downloads a 256 MB file at a fixed location, each node generates UDP traffic according to a two state Markov on-off process, with rates (per second) λ_{on} and λ_{off}. We fix $\lambda_{\text{on}} = 0.05$, and then perform experiments with $\lambda_{\text{off}} = 0.025$ and $\lambda_{\text{off}} = 0.05$. As in Section 4.2.3, we control background traffic for the WiFi channel only.

Figure 4.8 shows sample throughput traces of MPTCP and eMPTCP when two interfering nodes turn traffic on and off with $\lambda_{\text{on}} = 0.05$ and $\lambda_{\text{off}} = 0.025$. We observe that standard MPTCP avoids aggressive use of the LTE subflow when the WiFi subflow provides high bandwidth, e.g., during the interval 10–60 seconds in Figure 4.8.

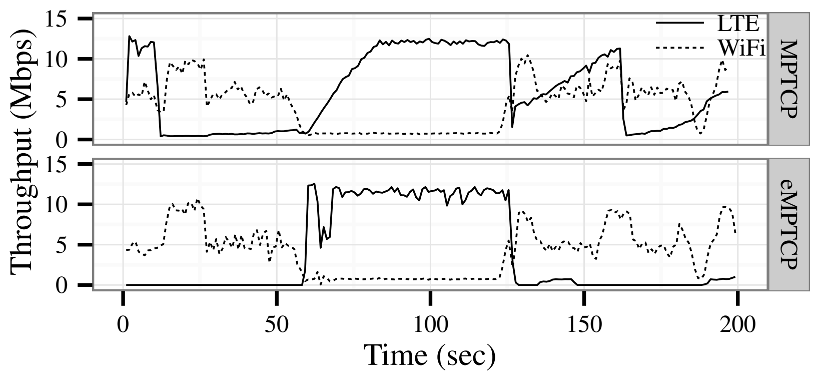


Figure 4.8. Example Throughput Trace with two WiFi Background Traffic Sources ($\lambda_{\text{on}} = 0.05$, and $\lambda_{\text{off}} = 0.025$)

This is because the MPTCP subflow scheduler chooses the WiFi subflow for packet transmission because it has a large CWND and the smallest RTT [63]. However, MPTCP consumes energy utilizing the LTE subflow even though the throughput gain is small. In contrast, eMPTCP suspends the LTE subflow when WiFi bandwidth is sufficiently large in order to avoid energy inefficient path usage. Note that at around time 60, the LTE throughput of MPTCP increases more slowly than that of eMPTCP. This is because the LTE subflow of MPTCP is in congestion avoidance after it experiences a loss at time 10 while that of eMPTCP is in slow-start since it is used for the first time. In Figure 4.8, eMPTCP incorrectly uses an LTE subflow in the intervals [140, 150] and [190, 200]. This is due to sudden decreasing WiFi throughputs, which result in incorrect throughput predictions. However, eMPTCP stops using the LTE subflow after the WiFi estimates improve.

Figure 4.9 presents the average energy consumption and download times for different values of n and λ_{off}, in percentage terms relative to MPTCP, i.e., smaller numbers are better, and percentages less than 100% are better than standard MPTCP. MPTCP's place on the Figure is denoted by the red dashed line. Values are averages over five experiments.

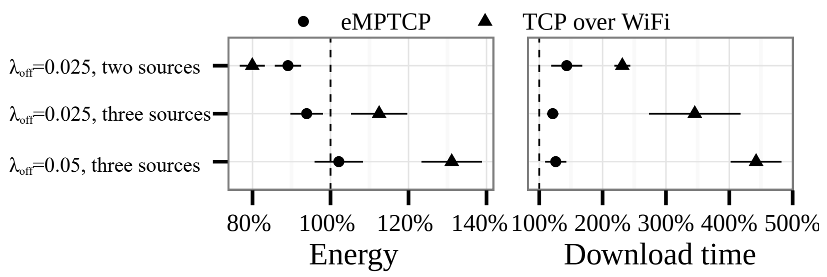


Figure 4.9. Comparison normalized relative to MPTCP in Random WiFi Background Traffic

Figure 4.9 shows that eMPTCP consumes less energy than MPTCP as the number of background sources and λ_{off} decreases. When $\lambda_{\text{off}} = 0.025$, we observe that the eMPTCP and WiFi-only are more efficient that MPTCP when there are two background sources. Recall that the energy efficiency of eMPTCP improves when it can suspend an LTE subflow in situations where using WiFi only is more energy efficient. Larger numbers of interfering WiFi nodes result in more losses caused by collisions when background traffic is present, resulting in more CWND decreases. Thus, the device is likely to obtain a larger TCP throughput with a larger CWND when there is no background traffic, resulting in a greater energy efficiency of TCP over WiFi and eMPTCP. Note that TCP over WiFi is most energy efficient when there are two background sources and $\lambda_{\text{off}} = 0.025$. However, as shown in Figure 4.9, in that setting, TCP over WiFi requires twice as much time to complete a download as eMPTCP, while it consumes just 11% less energy.

As shown in Figure 4.9, MPTCP provides the smallest download times, regardless of the number of background sources and λ_{off}, since it always utilizes the LTE subflow. In addition, as expected, the download time of TCP over WiFi becomes significantly larger as the number of background sources and λ_{off} increase. Compared to MPTCP, download times under eMPTCP are 20-40% larger while energy consumption is 9-11% lower. This may be because eMPTCP sometimes poorly predicts available bandwidth due to fluctuating throughputs, as illustrated in Figure 4.8,

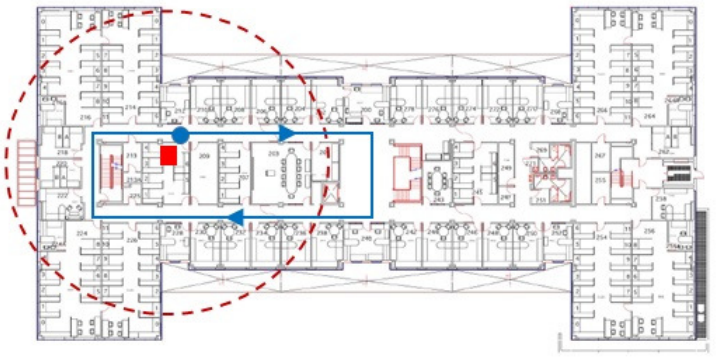


Figure 4.10. Mobile Scenario inside UMass CS building. Route starts at the blue point. The red square is the AP. The red dashed circle is the estimated usable access range of the AP.

and it also incurs additional energy overhead when suspending and resuming an LTE subflow due to promotion and tail costs. Compared to single-path TCP over WiFi, eMPTCP reduces download time by up to 70%, while at the same time using less energy.

4.2.5 Experiments with Mobility

The focus of this section is to determine how well eMPTCP performs and adapts in a mobile scenario. We take measurements while moving for 250 seconds along the route shown in Figure 4.10. To make our comparison between MPTCP and eMPTCP as fair as possible, we use the same route for the experiments.

Figure 4.11 presents sample traces of accumulated energy consumption from our mobile scenario. In this experiment, the device is sometimes within WiFi communication range, and sometimes outside it, depending on its location. As the device moves outside WiFi range, WiFi throughput decreases, e.g., the duration around 25-40 seconds in Figure 4.11. At the beginning of the route, MPTCP sestablishes both subflows, whereas eMPTCP postpones establishing an LTE subflow, since WiFi throughput is high enough to be more energy efficient than using both interfaces.

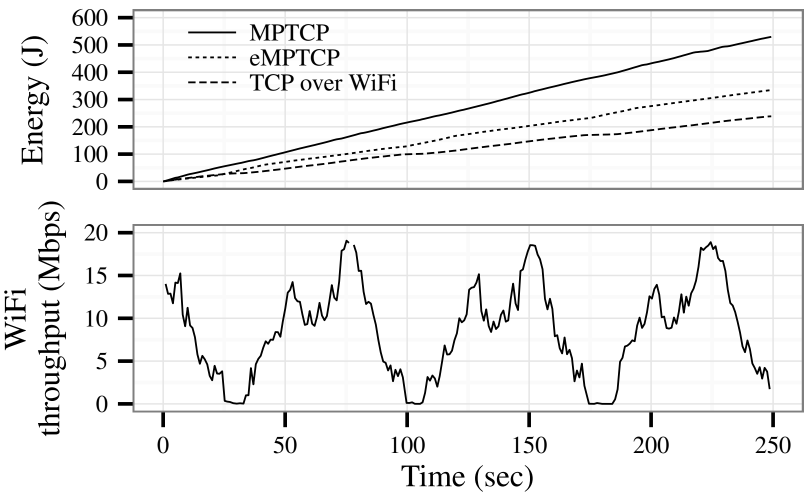


Figure 4.11. Example of Accumulated Energy Consumption with Mobile Scenario

However, eMPTCP establishes an LTE subflow after the WiFi bandwidth decreases when the device is leaving the AP communication range (at around 25 seconds in Figure 4.11). Then, whenever the device cannot obtain enough WiFi bandwidth to be more energy efficient than using both interfaces, eMPTCP utilizes the LTE subflow rather than only using the poorly performing WiFi subflow. In this experiment, we observe that since the device is inside WiFi communication range most of the time, eMPTCP utilizes the LTE subflow only for a few short time intervals. Therefore, as shown in Figure 4.11, the slope of eMPTCP's accumulated energy consumption (the energy consumption per second) is larger than that of TCP over WiFi, but smaller than that of MPTCP.

We now examine the per-byte energy efficiencies of MPTCP, eMPTCP, and single-path TCP over WiFi. Figure 4.12 compares the energy consumption per byte and download amount for 250 seconds averaged over five runs. eMPTCP's energy consumption per byte is 22% smaller than that of MPTCP and 15% larger than that of TCP over WiFi, since eMPTCP utilizes the LTE subflow for only several short periods. Because WiFi throughput degradation is due only to the distance between the AP and device (as there is no WiFi background traffic in these experiments), TCP

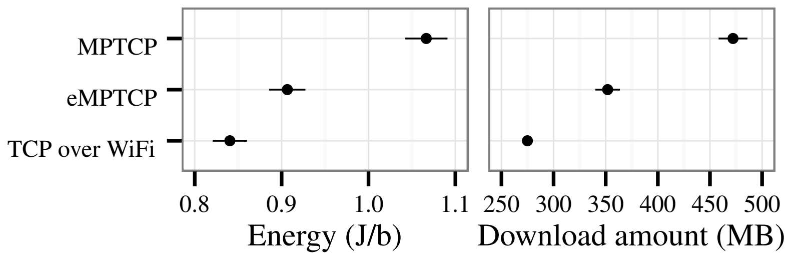


Figure 4.12. Mobile Scenario Comparison

over WiFi is slightly better in terms of energy efficiency than eMPTCP, corresponding to the case when there are two background sources and $\lambda_{\text{off}} = 0.025$ in Figure 4.9.

Focusing on the amount of data downloaded during the experiments, we observe that eMPTCP downloads 25% less data than MPTCP, while exhibiting 22% lower per-byte energy costs. Here, eMPTCP roughly loses about the same in performance that it saves in energy, due to the overhead of switching between WiFi-only and using both interfaces. eMPTCP downloads 28% more data than single-path TCP over WiFi even though it yields almost as good per-byte energy efficiency (just 8% more energy consumption per byte).

4.2.6 Comparison with existing approaches

Raiciu *et. al* [62] propose a simple strategy called MPTCP with WiFi First, where MPTCP only uses WiFi when available, and the cellular network otherwise. This is accomplished by placing the cellular subflow in backup mode and activating it only when WiFi is not available. This simple strategy may seem similar to eMPTCP, however, it cannot take advantage of dynamic situations where TCP over LTE or MPTCP is more energy-efficient than TCP over WiFi. MPTCP with WiFi First can only utilize an LTE subflow when the WiFi subflow explicitly breaks, such as due to a WiFi AP disassociation. For example, in our mobile scenario in Section 4.2.5, MPTCP with WiFi First would not use the LTE subflow even when the WiFi subflow becomes unusable, e.g., the duration around 25∼40 seconds in Figure 4.11, since the device does not lose the WiFi association. Therefore, if WiFi provides too low bandwidth to be more energy efficient than LTE while it is still associated, MPTCP with WiFi First degenerates into single-path TCP over WiFi, which yields inefficient energy usage as shown in subsections 4.2.2–4.2.5. It also needlessly activates the cellular interface at connection establishment.

Pluntke *et al.* [58] introduce a Markov Decision Process (MDP) based MPTCP path scheduler to minimize energy consumption. The MDP based scheduler cannot be computed in the OS Kernel at run time since it incurs a large computational overhead and a finite state machine of throughput changes with transition probabilities, which makes it hard to be practical in real deployments. Therefore, rather than directly implementing their approach in mobile devices, we generate the MDP based schedulers and simulate their behaviors given our experimental scenarios and energy model. Unit time for state transitions is set to one second as in [58]. Note that the authors consider only 3G and WiFi energy models in which 3G energy consumption becomes lower than WiFi for high data rates. In contrast, LTE energy consumption per *second* never becomes lower than WiFi in our energy model. We observe that the generated MDP schedulers choose WiFi-only for all scenarios, resulting in same energy performance (and limitations) as TCP over WiFi.

4.3 Evaluation in the Wild

We next examine whether eMPTCP provides greater energy efficiency than standard MPTCP in more realistic environments. We deploy MPTCP enabled servers in Asia (Singapore - SNG), Europe (Amsterdam - AMS), and North America (Washington D.C. - WDC). Each server has one network interface on the public Internet. We investigate the performance of eMPTCP when downloading files of different sizes (256 KB for small file transfer and 16 MB for large file transfer) at three locations: a university building where the WiFi AP is connected to the campus network, student

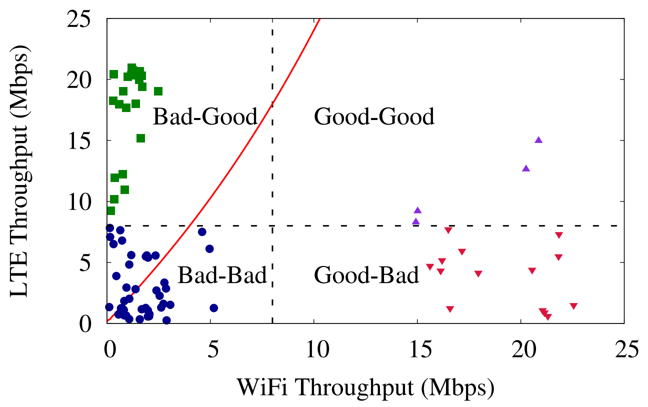


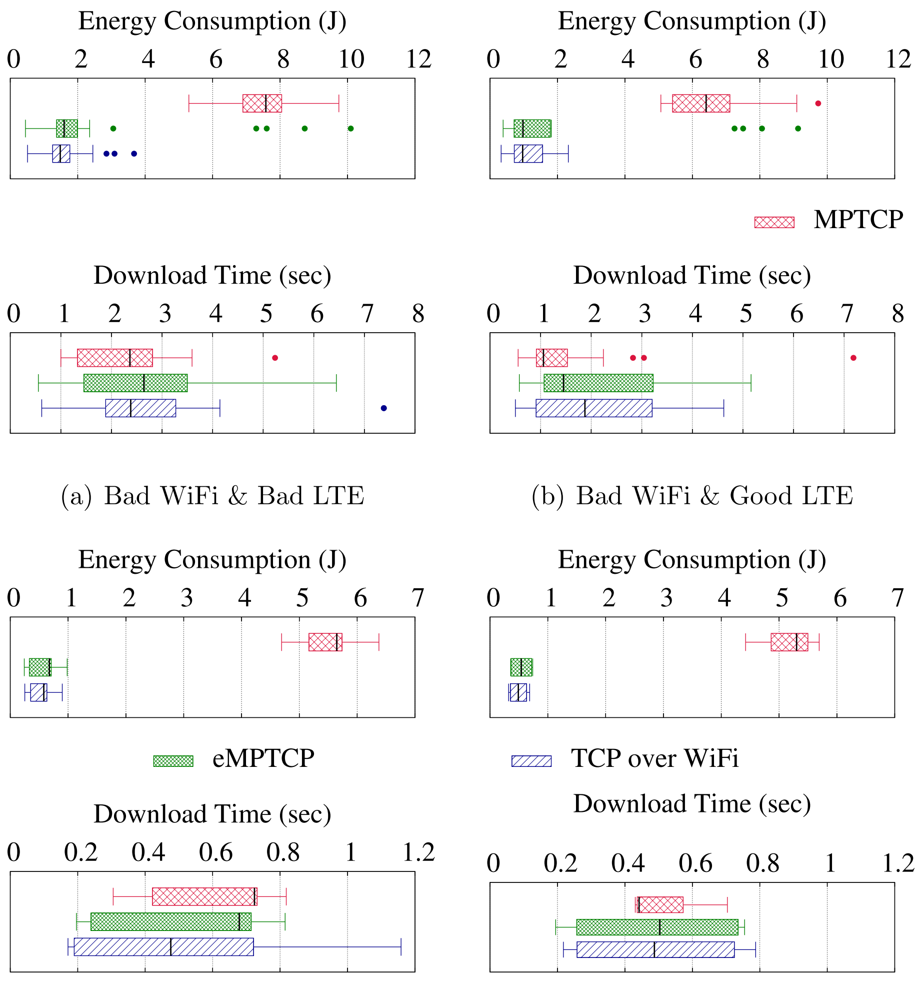
Figure 4.13. Trace Categories according to WiFi and LTE throughput - 16MB Downloads

housing where the AP is connected to the campus network through Cisco Long-Reach Ethernet, and a personal residence where the AP connects to a cable network.

We collect ten traces for each combination of file size, device and server locations. Since network conditions can vary over time, for each configuration we conduct ten iterations of a set of experiments. Each set consists of one run each of eMPTCP, original MPTCP, and single-path TCP over WiFi for the same configuration. The ordering within the set is randomized.

4.3.1 Trace Categorization

We group the collected traces into four categories based on the qualities of WiFi and LTE, either Good or Bad. We set 8 Mbps as a threshold to decide whether a network is good or bad. Figure 4.13 presents a scatterplot of measured WiFi and LTE throughputs for all experiments downloading 16 MB files. The scatterplot is divided into the four categories, and the red line indicates the boundary above which MPTCP is more energy efficient than TCP over WiFi for downloading.



(a) Bad WiFi & Bad LTE

(b) Bad WiFi & Good LTE

(c) Good WiFi & Bad LTE

(d) Good WiFi & Good LTE

Figure 4.14. Small File Transfers (256 KB Downloads)

4.3.2 Small File Transfers

Figure 4.14 presents Whisker plots showing first quartile (Q_1), median, third quartile (Q_3), and outliers of measured total energy consumptions and download times in the 256 KB traces for each of the four categories. Dots are outliers, which sit outside the range $[Q_1 - 1.5IQR, Q_3 + 1.5IQR]$, where IQR is the inter-quartile range defined as $(Q_3 - Q_1)$.

In this environment, eMPTCP almost always behaves the same as TCP over WiFi across all categories, yielding significantly less total energy consumption (from 75%

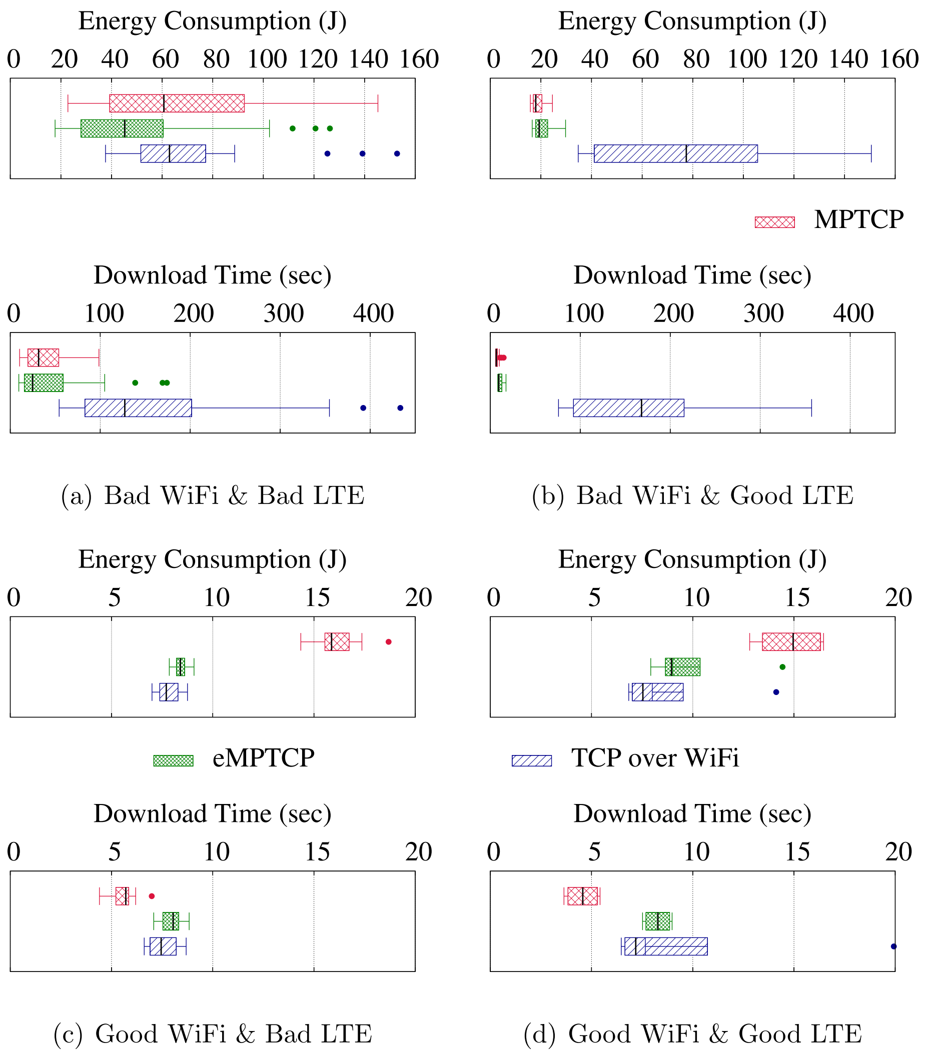


Figure 4.15. Large File Transfers (16 MB Downloads)

to 90%) with statistically similar download times as MPTCP. This is because the transfer is short, can be completed over WiFi, and eMPTCP avoids the tail state power costs of the LTE interface by delaying LTE subflow establishment.

eMPTCP does use the LTE subflow in a few instances, namely the outliers in Figure 4.14(a) and (b), which exhibit similar energy consumption to MPTCP. In these cases, LTE subflow establishment is triggered via timer expiration since the WiFi throughput obtained by eMPTCP is exceptionally small. Recall that after the timer expires, eMPTCP greedily chooses paths based on expected per-byte efficiency without knowing the length of the transfer remaining. Thus, even though the remaining data is too small for eMPTCP to take advantage of LTE's relative energy efficiency in this scenario, eMPTCP uses both because of the extremely slow WiFi.

4.3.3 Large File Transfers

Figure 4.15 presents Whisker plots of measured total energy consumption and download time of the 16 MB download experiments for each category. We observe the following:

Bad WiFi & Bad LTE: eMPTCP consumes 33% less energy than MPTCP and TCP over WiFi, while completing downloads in 20% less time. eMPTCP is the most energy efficient since it adaptively controls path usage according to expected per-byte efficiency. TCP over WiFi exhibits similar energy consumption to standard MPTCP but with roughly 6x longer download times.

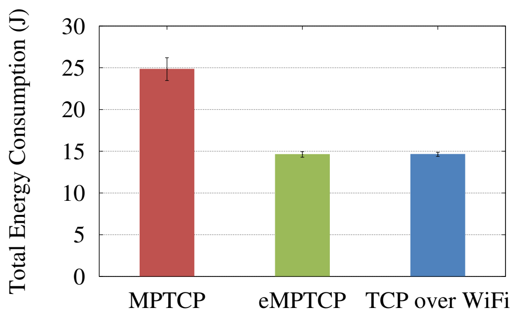
Bad WiFi & Good LTE: eMPTCP yields similar energy consumption to MPTCP with slightly larger download times. eMPTCP behaves similarly to MPTCP, since the throughput obtained over WiFi is small. The slightly larger download times than MPTCP are due to the delayed LTE subflow establishment. TCP over WiFi performs the worst, both in terms of download time and efficiency, since the WiFi is poor.

Good WiFi & Bad LTE: eMPTCP uses roughly 50% of the energy that MPTCP does, since it never utilizes the LTE subflow. TCP over WiFi behaves similarly. eMPTCP takes about 20% longer than MPTCP to complete downloads. eMPTCP essentially behaves the same as TCP over WiFi, and obtains similar results.

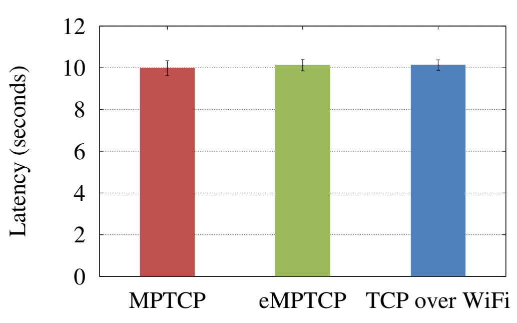
Good WiFi & Good LTE: This is similar to the above case, since the WiFi is fast and using WiFi-only is more power efficient than using both.

4.3.4 Case Study: Web Browsing

We now examine whether eMPTCP improves energy efficiency with a common application: Web browsing. To this end, we deploy a copy of CNN's home page



(a) Energy Consumption



(b) Latency

Figure 4.16. Web browsing Comparison

(as of 9/11/2014), which consists of 107 Web objects, into our MPTCP server in Washington DC. At our department building, we measure the energy consumption and latency to download all objects in the page. We consider the environment in this setting to have good WiFi and good LTE.

Figure 4.16 shows the average measured energy consumption and latency of MPTCP, eMPTCP, and single-path TCP over WiFi. In this experiment, the Android web browser establishes six parallel (MP)TCP connections to the server (12 subflows for MPTCP), with HTTP persistent connections. Note that the values are averaged over 10 runs and the power consumed for the Web browser application is included in the total energy consumption. As shown in Figure 4.16(a), MPTCP consumes 60% more energy (around 10J) than eMPTCP and TCP over WiFi. Since almost of all objects in the Web page are small (<256 KB), eMPTCP does not utilize the LTE network and achieves better energy efficiency. Figure 4.16(b) shows that eMPTCP yields almost the same latency as MPTCP with less energy consumption. This is because, for small downloads, it is hard for MPTCP to obtain gains by utilizing an LTE subflow in the case of small object downloads [7]. In contrast, eMPTCP automatically postpones an LTE subflow establishment unless a user initiates a large data transfer, such as video streaming. If a large transfer occurs, eMPTCP adaptively switches path usage between using both interfaces and WiFi-only, resulting in better energy efficiency than MPTCP. This example is meant to illustrate the potential benefits of eMPTCP for more complex workloads.

4.4 Related Work

Pluntke *et al.* [58] are the first to introduce the concept of scheduling paths in MPTCP to minimize energy consumption. They use a scheduler based on a Markov decision process. Schedules are expensive to compute, hence they are computed in the cloud and downloaded periodically to the device, which is impractical for real deployment. They evaluate their scheduler via simulation, using models of device energy consumption. They find they can reduce energy consumption by almost 10% in one out of four scenarios. Our algorithm, in contrast, is evaluated experimentally using a real MPTCP implementation on a physical device, and across many scenarios. It also achieves much higher energy reductions, up to 90%. Pluntke *et al.* [58] is discussed in more detail in Section 4.2.6.

Raiciu *et al.* [62] consider several issues that arise when using MPTCP in mobile environment, including energy consumption. They propose and evaluate a simple strategy that periodically samples both paths for 10 seconds and then uses the more energy-efficient path for 100 seconds. Evaluating their approach via simulation, their method is more energy efficient than an energy-unaware MPTCP implementation, but achieves lower bandwidths. They also propose a strategy called "MPTCP with WiFi First", which exclusively uses WiFi when it is present, and only uses the cellular network when WiFi is unavailable. However, this approach cannot avoid inefficient use of energy due to automatic activation of the cellular interface and the attendant promotion and tail state costs. Raiciu *et al.* [62] are discussed in more detail in Section 4.2.6.

Paasch *et al.* [52] study mobile/WiFi handover performance with MPTCP. They suggest *Single-Path Mode*, which establishes a new subflow only after the current interface goes down. By setting WiFi as the primary interface, Single-Path Mode MPTCP can avoid the fixed energy overhead for 3G/LTE when WiFi is available.

Compared to our approach, however, both strategies are unable to take advantage of more dynamic situations where TCP over LTE or MPTCP may become more energy efficient than TCP over WiFi. In addition, they can only utilize an LTE subflow when the WiFi subflow explicitly breaks, such as disassociating with an AP.

Peng *et al.* [57] study energy consumption in MPTCP via analysis as a global optimization problem. They present two algorithms, customized for real-time and file transfer, and evaluate them via simulation. While they do utilize an energy model, they do not consider cellular promotion and tail costs, and do not evaluate their approach experimentally.

Bui *et al.* [4] present GreenBag, a middleware system to aggregate bandwidth of asymmetric wireless links for video streaming. GreenBag estimates the available bandwidth of WiFi and LTE and determines the amount of traffic to allocate to each interface. The authors show that GreenBag reduces energy consumption by 14∼25% compared with a bandwidth aggregation for throughput maximization. Since GreenBag operates above the TCP layer as a system background process, it only works for modified HTTP requests sent not to original destinations but to GreenBag. Thus, each application needs to be modified to cooperate with GreenBag, making deployment difficult. MPTCP, in contrast, requires no application modifications and works with all TCP traffic.

Ding *et al.* [16] proposes a mobile traffic offloading architecture that utilizes WiFi to migrate mobile traffic from cellular while reducing energy consumption. The authors demonstrate that the proposed architecture obtains more than 80% energy savings, using a prototype on a Nokia N900 smartphone and their own music streaming application. However, they do not consider potential energy gain from simultaneous use of both interfaces. As with Bui *et al.* [4], applications must be modified for the approach to work.

Croitoru *et al.* [11] show that MPTCP can improve user experience over WiFi by associating with multiple APs simultaneously. By utilizing an MPTCP connection with subflows connected to all available APs, a mobile client can maintain seamless connectivity without having to consider a handover. The authors also investigate situations when connecting to multiple APs can degrade network performance. They propose estimating client-side AP efficiency and using ECN notification to direct traffic to subflows associated with the most efficient APs while avoiding use of subflows that experience poor throughput. Their study does not consider cellular usage or energy consumption.

4.5 Conclusion

This chapter proposed, implemented, and evaluated an enhanced MPTCP designed to improve energy efficiency, called eMPTCP. eMPTCP manages subflow usage based on expected per-byte energy efficiency given available bandwidth. Our experimental results using our implementation in real mobile devices show that eMPTCP is able to consume less energy than MPTCP while still providing MPTCP's benefits of multi-path such as improved performance, availability, and transparency.

CHAPTER 5

MPTCP PATH SCHEDULING TO MANAGE PATH ASYMMETRIES

One significant component that affects MPTCP performance is the path scheduler, which distributes traffic across available paths according to a particular scheduling policy. The default path scheduler of MPTCP is based on round trip time (RTT) estimates, that is, given two paths with available congestion window space, it sends traffic over the path with the smaller RTT. While simple and intuitive, this scheduling policy does not carefully consider path heterogeneity, which can cause significant reorderings at the receiver-side. To prevent this, MPTCP includes opportunistic retransmission and penalization mechanisms along with the default scheduler [63]. When flows are long-lived, e.g., a single very large file transfer, MPTCP is able to enhance performance using these mechanisms. However, a large number of Internet applications such as Web browsing and video streaming usually generate traffic that consist of multiple uploads/downloads for relatively short durations. We find that in the presence of path asymmetries, the default MPTCP scheduler is unable to efficiently utilize some paths with such a traffic pattern. In particular it does not fully utilize fast paths, which have short RTT with high bandwidth and should be prioritized to achieve the highest throughput and lowest response time.

In this work, we propose a novel MPTCP path scheduler to maximize path utilization, called ECF (Earliest Completion First). ECF monitors not only RTT estimates, but also the current subflow congestion windows and the amount of data available to send (i.e., the send buffer). By determining whether using a slower path for the injected traffic will cause the faster path to become idle, ECF more efficiently utilizes the faster paths, maximizing throughput, minimizing download times, and reducing out-of-order packet delivery. Our experimental results demonstrate that ECF successfully avoids undesirable idle periods, achieving greater throughput with higher path utilization than the default scheduler.

This chapter makes the following contributions:

- We provide a thorough analysis of the performance problems in MPTCP caused by path asymmetries when using the default scheduler. Using a streaming adaptive bit rate video workload, we illustrate how it does not achieve the ideal use of multiple paths.

- Based on this insight, we design and implement a new path scheduler, Earliest Completion First (ECF) that takes path asymmetries into account.

- We evaluate ECF against the default scheduler and two other approaches, BLEST and DAPS, on a experimental testbed. We use workloads of streaming, simple downloads, and a full Web page download, across paths with a range of bandwidths and round-trip times. We show how ECF improves performance by up to 26% above the other schedulers in asymmetric path environments, while doing no worse in symmetric ones.

5.1 Motivation

5.1.1 Dynamic Adaptive Streaming over HTTP

Dynamic Adaptive Streaming over HTTP (DASH) [72] is the mechanism by which most video is delivered over the Internet. To stream videos with a bit rate appropriate for the available bandwidth, a DASH server provides multiple representations of a video content encoded at different bit rates. Each representation is fragmented into small video chunks that contain several seconds of video. Based on measured available

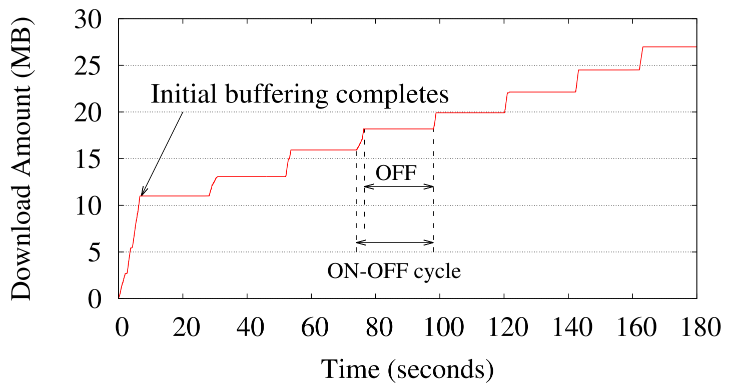


Figure 5.1. Example Download Behavior in Netflix Player

bandwidth, a DASH client selects a chunk representation, i.e., bit rate, and requests it from a DASH server; this is called adaptive bit rate (ABR) selection.

A DASH client player starts a streaming session with an initial buffering phase during which the player fills its playback buffer to some prescribed maximum level. During this phase, once the buffer reaches a second sufficient threshold, the player starts playing the video, and continues to retrieve video chunks until the initial buffering completes. After completing the initial buffering phase, the player pauses video download until the buffer level falls below the prescribed maximum level. If the playback buffer level falls below a prescribed minimum required to play out the video, the player stops playback and fills its buffer until it has a sufficient amount of video to begin playback again, which is called the rebuffering phase.

This can lead to an ON-OFF traffic pattern where the player downloads chunks for a period of time and then waits until a specific number of chunks are consumed [64]. Figure 5.1 shows an example of client player download behavior when a mobile device fetches Netflix streaming video. This trace was collected using an Android mobile handset (Samsung Galaxy S3) while watching Netflix through WiFi on May 2014. During the OFF periods, the connection can go idle, causing CWND resets, as we will discuss in Section 5.1.3.

Resolution	144p	240p	360p	480p	760p	1080p
Bit Rate (Mbps)	0.26	0.64	1.00	1.60	4.14	8.47

Table 5.1. Video Bit Rates vs. Resolution

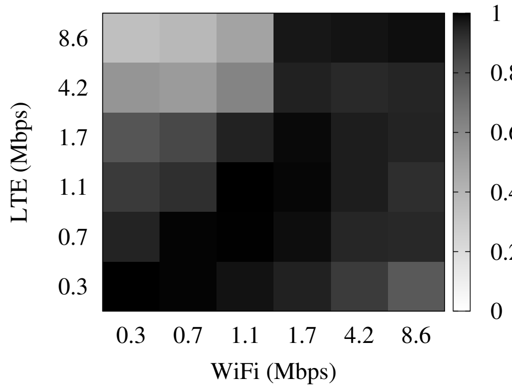


Figure 5.2. Measured Average Bit Rate Relative to Ideal Average Bit Rate Using Default Path Scheduler in MPTCP (darker is better)

5.1.2 The Effect of Path Asymmetry

We first examine the effect of path asymmetry on MPTCP application performance using adaptive video streaming, since it is currently one of the dominant applications in use over the Internet. We measure the average video bit rate obtained by an Android DASH streaming client while limiting the bandwidth of the WiFi and LTE subflows on the server-side using the Linux traffic control utility tc [43] (full details of our experimental setup are given in Section 5.4.1). The streaming client uses state-of-art adaptive bit rate selection (ABR) [31]. The choice of ABR does not significantly affect the results as we fix bandwidth on each interface for the duration of each experiment.

Table 5.1 presents the bit rates corresponding to each resolution. We choose bandwidth amounts slightly larger than those listed in Table 5.1, i.e., $\{0.3, 0.7, 1.1, 1.7, 4.2, 8.6\}$ Mbps, to ensure there is sufficient bandwidth for that video encoding.

Figure 5.2 presents the ratio of the average bit rate achieved versus the ideal average bit rate available, based on the bandwidth combinations, when using the default MPTCP path scheduler. The figure is a grey-scale heat map where the darker the area is, the closer to the ideal bit rate the streaming client experiences. The closer the ratio is to one, the better the scheduler does in achieving the potential available bandwidth. The values are averaged over five runs. In a streaming workload, we define the ideal average bit rate as the minimum of the aggregate total bandwidth and the bandwidth required for the highest resolution at that bandwidth. For example, in the 8.6 Mbps WiFi and 8.6 Mbps LTE pair (the upper right corner in Figure 5.2), the ideal average bit rate is 8.47 Mbps, since the ideal aggregate bandwidth ($8.6 + 8.6 = 17.2$ Mbps) is larger than the required bandwidth for the highest resolution of 1080p (8.47 Mbps). Since the full bit rate is achieved, the value is one and the square is black.

Figure 5.2 shows that, when significant path asymmetries exist, the streaming client fails to obtain the ideal bit rate. For example, when WiFi and LTE provide 0.3 Mbps and 8.6 Mbps, respectively (the upper left box in Figure 5.2), the streaming client retrieves 480p video chunks, which requires only 2 Mbps, even though the ideal aggregate bandwidth is larger than 8.47 Mbps. Thus, the value is only 25% of the ideal bandwidth and the square is light grey. This problem becomes even more severe when the primary path (WiFi) becomes slower (compare the 0.3 Mbps & [0.3 – 8.6] Mbps and 8.6 Mbps & [0.3 – 8.6Mbps] pairs), as shown by the grey areas in the upper left and lower right corners.

Note that we observe similar performance degradation regardless of the congestion controller used (e.g., Olia [37]). In addition, the opportunistic retransmission and penalization mechanisms are enabled by default. This result shows that even with these mechanisms, the MPTCP default path scheduler does not sufficiently utilize the faster subflow when path asymmetries are present.

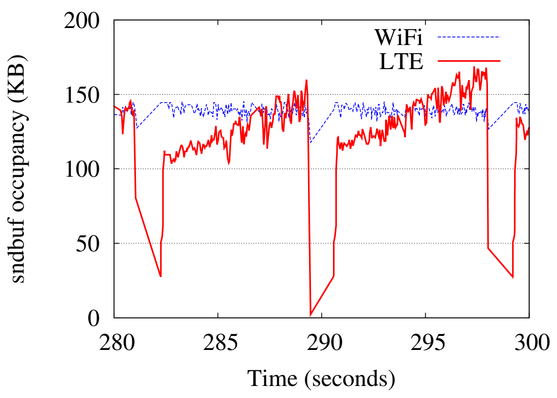


Figure 5.3. Send Buffer Occupancy (0.3 Mbps WiFi and 8.6 Mbps LTE. Including in-flight packets)

5.1.3 Why Does Performance Degrade?

In this section, we identify the cause of the performance degradation when path asymmetries exist. We investigate the TCP send buffer behavior of the faster subflow in the traces of the streaming experiments. Figure 5.3 shows the send buffer occupancy (measured in the kernel) of the WiFi and LTE subflows when bandwidths are 0.3 and 8.6 Mbps, respectively. As can be seen, the streaming sender application periodically pauses to queue data into the LTE subflow, which has significantly higher bandwidth and lower RTT than the 0.3 Mbps WiFi subflow, and the LTE send buffer quickly empties due to acknowledgements. The streaming sender also pauses to use the WiFi subflow, i.e., the sender has no packet to send, but the sender is still transferring data over the slow WiFi subflow while the fast LTE subflow is idle. This shows that the application does not have any packet to send at that moment; the 8.6 Mbps LTE subflow completes its assigned packet transmissions much earlier than the 0.3 Mbps WiFi subflow and stays idle until the next download request is received.

Figure 5.4 presents a timing diagram to show how a fast subflow becomes idle, waiting until a slow subflow completes assigned packet transmissions (here, subflow 1 is faster than subflow 2). To validate whether such an idle period really occurs, we investigate the CDF of the time difference between the last packets over WiFi and

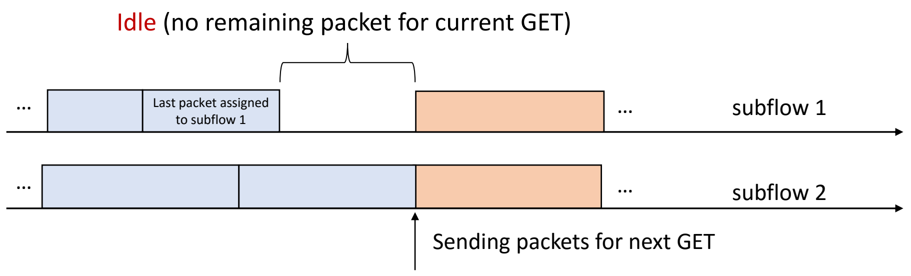

Idle (no remaining packet for current GET)
Last packet assigned to subflow 1
subflow 1
subflow 2
Sending packets for next GET

Figure 5.4. Case When Fast Subflow Becomes Idle

LTE for four regulated bandwidth pairs. As shown in Figure 5.5, as paths become more heterogeneous, the time differences increases. In particular, the pause period (around one second) in Figure 5.3 appears as the time difference of last packets. Note that this problem is due to the lack of packets to send, and not because of head of line blocking or receive window limitation problems discussed in [63]. The performance degradation of these idle periods becomes more severe as an MPTCP connection is used for more multiple object downloads, e.g. a streaming client continues to downloads video chunks over the connection. This is because the congestion controller restarts from the slow-start phase with CWND reset if a connection is idle for longer than the retransmission timeout [27]. Since MPTCP congestion controllers such as coupled and Olia are designed to adapt a subflow CWND as a function of CWNDs across all subflows, resetting CWND in a fast subflow due to an idle period can result in it not being properly increased, resulting in under-utilization of the fast subflow for consecutive downloads.

Simple scheduling policies based solely on RTTs, e.g., allocating traffic to each subflow inversely proportional to RTT [39], cannot prevent this problem. For example, consider two subflows where the RTTs are 10 ms and 100 ms, respectively, and the CWNDs of both subflows are 10 packets. Suppose that the sender has 11 packets remaining to transmit. If a scheduler splits these 11 packets based on RTT, the fast subflow will complete 10 packet transmissions in one RTT (10 ms) and the slow

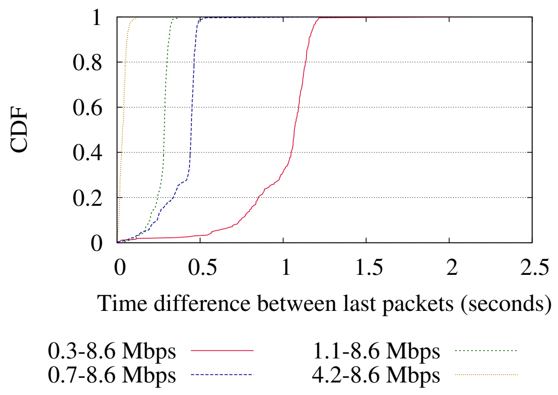


Figure 5.5. Time Difference of Last Packets

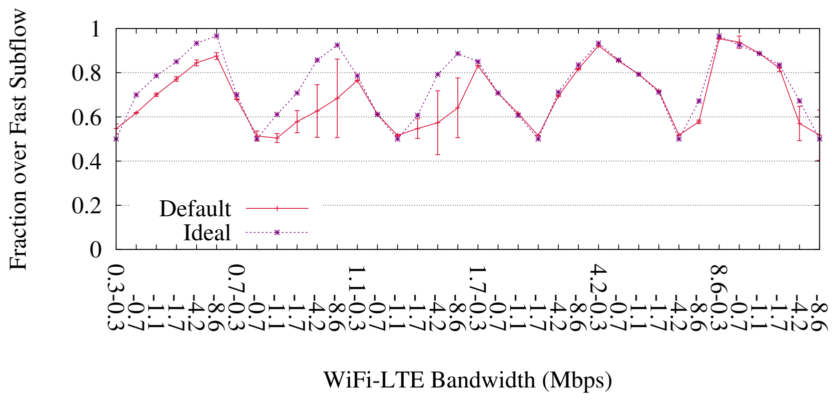


Figure 5.6. Fraction of Traffic Allocated to Fast Subflow using Default Scheduler in Streaming

subflow one packet in 100 ms. This results in a completion time of 100 ms, where the faster subflow is idle for 90 ms. In contrast, waiting for the 10 ms subflow to become available results in a completion time of just 20 ms.

An ideal MPTCP path scheduler is one that splits traffic in proportion to the available bandwidth of each path. Figure 5.6 presents the average fraction of traffic allocated to the fast subflow during the streaming experiments. As can be observed, the default scheduler places a smaller fraction of the traffic onto the fast subflow than the ideal scheduler. Together with the idle period of the fast subflow, this causes the aggregate throughput degradation, resulting in a lower streaming quality selection than is possible given the available bandwidth.

5.2 Earliest Completion First Scheduler

To solve the performance degradation problem with path asymmetry, we propose a new MPTCP path scheduler, called ECF (Earliest Completion First). ECF utilizes RTT estimates, path bandwidths (in the form of congestion window sizes), and the size of the send buffer at the connection-level.

An MPTCP sender stores packets both in its connection-level send buffer and in the subflow level send buffer (if the packet is assigned to that subflow). This means that if the number of packets in the connection level send buffer is larger than the aggregate number of packets in the subflow level send buffers, there are packets in the send buffer that need to be scheduled to the subflows.

Assume that there are k packets in the connection level send buffer, which have not been assigned (scheduled) to any subflow. If the fastest subflow in terms of RTT has available CWND, the packet can simply be scheduled to that subflow. If the fastest subflow does not have available space, the packet needs to be scheduled to the second fastest subflow.

We denote the fastest and the second fastest subflows as x_f and x_s, respectively. Let RTT_f, RTT_s and $CWND_f$, $CWND_s$ be the RTTs and CWNDs of x_f and x_s, respectively. If the sender waits until x_f becomes available and then transfers k packets through x_f, it will take approximately $RTT_f + \frac{k}{CWND_f} \times RTT_f$, i.e., the waiting and transmission time of k packets. Otherwise, if the sender sends some packets over x_s, the transmission will finish after RTT_s with or without completing k packet transfers. Thus, as shown in Figure 5.7, in the case of $RTT_f + \frac{k}{CWND_f} \times RTT_f < RTT_s$, using x_f after it becomes available can complete the transmission earlier than using x_s at that moment. If $RTT_f + \frac{k}{CWND_f} \times RTT_f \geq RTT_s$, there are sufficient number

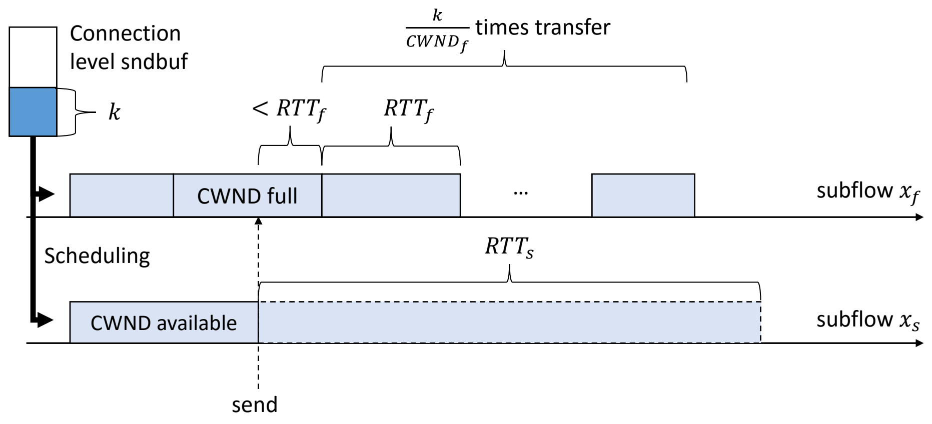


Figure 5.7. The case for waiting for the fast subflow

Algorithm 2 ECF Scheduler

```
// This function returns a subflow for packet transmission
Find fastest subflow $x_f$ with smallest $RTT$
if $x_f$ is available for packet transfer then
    return $x_f$
else
    Select $x_s$ using MPTCP default scheduler
    $n = 1 + \frac{k}{CWND_f}$
    $\delta = \max(\sigma_f, \sigma_s)$
    if $n \times RTT_f < (1 + waiting \times \beta)(RTT_s + \delta)$ then
        if $\frac{k}{CWND_s} \times RTT_s \geq 2RTT_f + \delta$ then
            // Wait for $x_f$
            $waiting = 1$
            return no available subflow
        else
            return $x_s$
        end if
    else
        $waiting = 0$
        return $x_s$
    end if
end if
```

of packets to send, so that using x_s at that moment can decrease the transmission time by utilizing more bandwidth than just by using x_f.

Based on this idea, we devise the ECF (Earliest Completion First) scheduler. Algorithm 2 presents the pseudo code for ECF.

Note that the inequality uses RTT estimates and CWND values, which can vary over time. To compensate for this variability, we add a margin $\delta = \max(\sigma_f, \sigma_s)$, where σ_f and σ_s are the deviations of RTT_f and RTT_s, respectively, in the inequality for the scheduling decision,

$$\left(1 + \frac{k}{CWND_f}\right) \times RTT_f < RTT_s + \delta$$

This inequality takes into account the case in Figure 5.7, in which waiting for the fastest subflow completes transfer earlier than using the second fastest subflow. To more strictly assure this case, ECF checks an additional inequality, which validates if using the second fastest subflow with its CWND (it takes $\frac{k}{CWND_s} \times RTT_s$ to finish transfer) does not complete earlier than waiting for the fastest subflow (at least $2RTT_f$ for transfer),

$$\frac{k}{CWND_s} \times RTT_s \geq 2RTT_f + \delta$$

Here, we also use δ to compensate for RTT and CWND variabilities.

If these inequalities are satisfied, ECF does not use the second fastest subflow x_s and instead waits for the fastest subflow x_f to become available. ECF uses a different inequality for switching back to using x_s after deciding to wait for x_f:

$$\left(1 + \frac{k}{CWND_f}\right) \times RTT_f < (1 + \beta)(RTT_s + \delta).$$

This adds some hysteresis to the system and prevents it from switching states (waiting for x_f or using x_s now) too frequently.

ECF can be adapted to more than two subflows, although it compares only two subflows, x_f and x_s at every scheduling decision, the outcome determined by the following proposition.

Proposition 1. *If ECF decides to wait for the fastest subflow x_f rather than using x_s, using other available subflows cannot contribute to earlier transfer completion.*

Proof. x_f is expected to be available in RTT_f, which is the smallest, and completes transfer in $\frac{k}{CWND_f} \times RTT_f$. The other subflows x_i not selected as x_s have a larger RTT (RTT_i) than RTT_s. Note that ECF waits for x_f if it results in an earlier transfer completion than RTT_s. Any packet transfer over x_i takes at least RTT_i, which is larger than RTT_s and RTT_f. Therefore, any other subflow x_i not considered as x_s must satisfy the inequality $RTT_f + \frac{k}{CWND_f} \times RTT_f < RTT_s < RTT_i$ that x_s does. This means that using x_i at that moment cannot shorten the transfer completion time compared to waiting for x_f, which shows that ECF yields optimal decisions. □

5.3 Implementation

We implement the ECF scheduler in the Linux Kernel using MPTCP code revision 0.89 from [51]. To obtain the required information for ECF, we utilize the smoothed mean and deviation of the RTT estimates and the send buffer information in the standard TCP kernel implementation.

MPTCP uses two types of sockets to manage a connection at connection and subflow levels. Therefore, by comparing the send buffer information between the connection and subflow sockets, we can estimate the number of packets k in the connection-level send buffer not assigned to subflows. We exploit the `sk_wmem_queued` field in the `struct sock`, which is the number of bytes queued in the socket send buffer that either have not been sent, or not been acknowledged. By subtracting the sum of `sk_wmem_queued` of the subflow sockets from that of the connection socket, we can estimate the number of bytes not yet allocated to the subflows. However, MPTCP preserves packets in the connection-level send buffer unless those packets are acknowledged at the connection level. That is, the number of in-flight packets in the connection socket can be larger than the sum of in-flight packets in the subflow sockets. Therefore, to prevent cases where packets acknowledged at only subflow-level are counted as packets not assigned to the subflows, we subtract the number of bytes in the connection socket that are already acknowledged in the subflow sockets. To this end, we utilize the `packets_out` field in `struct tcp_sock`, which is the number of in-flight packets of the TCP socket. Since `packets_out` is denominated in packets, not bytes, we assume that each packet has the same size, that of the maximum segment size (MSS) of the socket.

Let `meta_sk` and $\texttt{sk}_i$ denote the connection and subflow i sockets, respectively. Denote the TCP sockets corresponding to `meta_sk` and $\texttt{sk}_i$ by `meta_tp` and $\texttt{tp}_i$, i.e., $\texttt{meta\_tp} = \texttt{tcp\_sk(meta\_sk)}$ and $\texttt{tp}_i = \texttt{tcp\_sk(sk}_i\texttt{)}$. Then we estimate k (in bytes) as follows:

$$\begin{aligned} k \;=\;& \texttt{meta\_sk->sk\_wmem\_queued} - \\ & \sum_i \texttt{sk}_i\texttt{->sk\_wmem\_queued} - \\ & \left(\texttt{meta\_tp->packets\_out} - \sum_i \texttt{tp}_i\texttt{->packets\_out}\right) \times \texttt{MSS} \end{aligned}$$

To collect RTT estimates, we use `srtt` and `rttvar` in `struct tcp_sock`[1], which are the smoothed round trip time and maximal deviation for the last RTT periods, respectively, i.e., $RTT_i = \texttt{tp}_i\texttt{->srtt}$ and $\sigma_i = \texttt{tp}_i\texttt{->rttvar}$.

To estimate CWND, we utilize `snd_cwnd` in `struct tcp_sock`, which is the CWND measured in packets. We assume that a scheduling decision usually happens after a congestion controller enters congestion avoidance phase and, thus, use the value of `snd_cwnd` at that moment for evaluating the inequality in the algorithm. However, streaming ON-OFF traffic pattern or incorrect scheduling decision can cause a sub-

[1]Note that our implementation is based on MPTCP 0.89 forked from Kernel 3.14.33, in which RTT estimates are in jiffies. More recent Kernel like 3.18.34 maintains RTT estimates in terms of microseconds, e.g., `srtt_ms`.

flow or connection to idle, which can trigger a CWND reset when the idle period is longer than the retransmission timeout in RFC2861 [27]. When the CWND is reset on the subflow, ECF will use an unnecessarily small CWND, and will go through slow-start. To avoid this behavior, ECF records the largest value of `snd_cwnd` right before a CWND idle reset (`rec_snd_cwnd`) event. ECF resets `rec_snd_cwnd` to zero if the current `snd_cwnd` becomes larger than `rec_snd_cwnd`. ECF uses the maximum of current `snd_cwnd` and `rec_snd_cwnd` for $CWND$ as:

$$CWND = \max\left(\texttt{tp->snd\_cwnd}, \texttt{tp->rec\_snd\_cwnd}\right).$$

Note that this CWND value is used only for ECF decisions; ECF does not change the current CWNDs that the congestion controller uses (`tp->snd_cwnd`). Thus, our actions are consistent with RFC 2861.

The source code of the ECF scheduler is available at `http://cs.umass.edu/~ylim/mptcp_ecf`.

5.4 Evaluation in a Controlled Lab

5.4.1 Experimental Setup

In our lab setting, we examine performance using three workloads: adaptive streaming video over HTTP, simple download activity using wget, and Web-browsing.

We use an Android mobile device (Google Nexus 5) as the client. Videos are played on the device using ExoPlayer [24]. The mobile device communicates with the server over the Internet using a WiFi access point (IEEE 802.11g) and an LTE cellular interface from AT&T. Note that MPTCP requires a default primary interface with which to initiate and receive transfers. While the choice of interface to use as the primary is a complex one [14], we use WiFi as the primary interface since that is the default in Android.

For the server, we use a desktop running Ubuntu Linux 12.04 with the MPTCP 0.89 implementation deployed [51]. The server has an Intel Core I7-3770 CPU, 32 GB of memory, and is connected to our campus network through a single Gigabit Ethernet interface. The opportunistic retransmission and penalization mechanisms are enabled throughout all experiments.

On the server, we set up an MPTCP streaming server that provides DASH content as well as Web pages. We use Apache 2.2.22 as the HTTP server while enabling HTTP persistent connections with the default Keep Alive Timeout (5 sec).

For DASH content, we select a video clip from [33] that is 1332 seconds long and encoded at 50 Mbps by an H.264/MPEG-4 AVC codec. The original resolution of the video is 2160p (3840 by 2160 pixels). We configure the streaming server to provide six representations of the video with resolutions varying from 144p to 1080p (just as Youtube does). We re-encode the video file at each resolution and create DASH representations with 5 second chunks. Recall Table 5.1 in Section 5.1 presents the bit rates corresponding to each resolution.

The ECF hysteresis value β is set to 0.25 throughout our experiments (other values for β were examined but found to yield similar results, not shown due to space limitations). We compare ECF to the following schedulers:[2]

- Default: The default scheduler allocates traffic to a subflow with the smallest RTT and available CWND space. If the subflow with the smallest RTT does not have available CWND space, it chooses an available subflow with the second smallest RTT.

- Delay-Aware Packet Scheduler (DAPS) [39]: DAPS seeks in-order packet arrivals at the receiver by deciding the path over which to send each packet based

[2]For DAPS and BLEST, we use the implementation from `https://bitbucket.org/blest_mptcp/nicta_mptcp` [18]

on the forward delay and CWND of each subflow: DAPS assigns traffic to each subflow inversely proportional to RTT.

- Blocking Estimation-based Scheduler (BLEST) [18]: BLEST aims to avoid out-of-order delivery caused by sender-side blocking when there is insufficient space in the MPTCP connection-level send window. When this send window is mostly filled with packets over a slow subflow, the window does not have enough space, and the sender cannot queue packets to an MPTCP connection. To avoid this situation, BLEST waits for a fast subflow to become available, so that the fast subflow can transmit more packets during the slow subflow's RTT, so as to free up space of the connection-level send window.

BLEST and ECF are similar in that both can decline opportunities to send on the slow subflow when it has available CWND space, but this decision is based on different design goals. BLEST's decision is based on the space in MPTCP send window and minimizing out-of-order delivery, whereas ECF's is based on the amount of data queued in the send buffer and with the goal to minimize completion time. We will show in Section 5.4.2.3 that ECF better preserves the faster flow's CWND and thus performs better.

5.4.2 Video Streaming with Fixed Bandwidth

We begin by investigating whether ECF improves the performance of streaming applications compared to the other schedulers, keeping bandwidth fixed for the duration of the experiment.

5.4.2.1 Measured Bit Rate

We first compare how the schedulers perform using our streaming workload. Figure 5.8 presents the relative average bit rate of the default, ECF, DAPS and BLEST schedulers, normalized by the ideal average bit rate. The numbers are averaged over

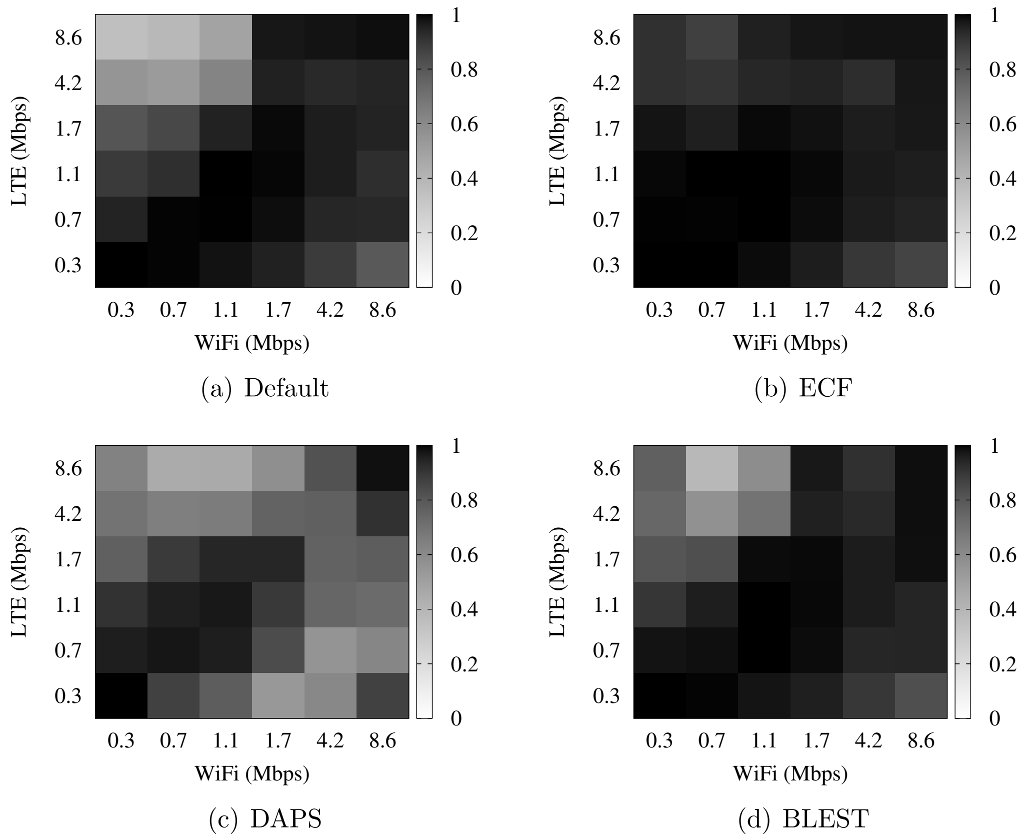


Figure 5.8. Measured Average Bit Rate Relative to Ideal Average Bit Rate (darker is better)

Bandwidth (Mbps)	0.3	0.7	1.1	1.7	4.2	8.6
WiFi RTT(ms)	969	413	273	196	87	40
LTE RTT(ms)	858	416	268	210	131	105

Table 5.2. Avg. RTT with Bandwidth Regulation

three runs. Table 5.2 presents the average RTT of each interface measured at sender-side according to the bandwidth configurations. Note that with the same bandwidth regulation, WiFi yields smaller RTTs than LTE, since the WiFi network is located in our campus network and is closer (measured in number of hops) to our client than over the AT&T LTE network.

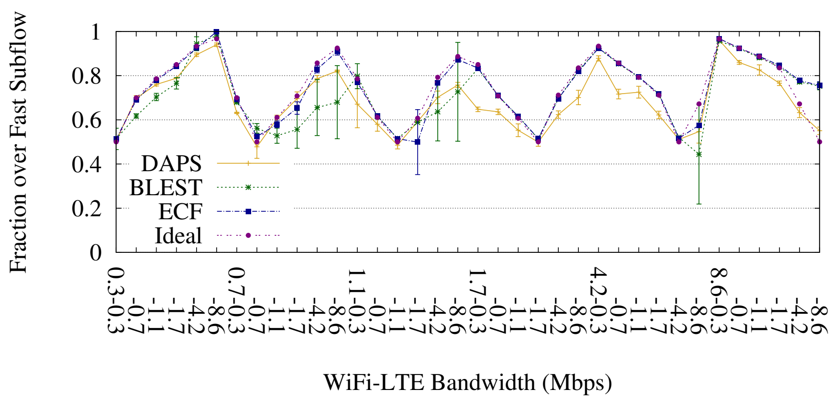


Figure 5.9. Fraction of Traffic Allocated to Fast Subflow in Streaming Workload - Fixed Bandwidth

Figure 5.8(b) shows that ECF successfully enables the streaming client to obtain average bit rates closest to the ideal average bit rate, and does substantially better than the default when paths are not symmetric.

Comparing Figure 5.8(c) to Figure 5.8(a), DAPS does not improve streaming performance; it yields even worse streaming bit rate than the default scheduler with some bandwidth configurations, e.g., 4.2Mbps for both of WiFi and LTE. Comparing Figure 5.8(d) with Figure 5.8(a), BLEST slightly improves streaming performance with 1 Mbps WiFi and [1..10] Mbps LTE pairs, but does not improve the average bit rate for other configurations.

5.4.2.2 Traffic Split

To understand why ECF performs better, we examine how each scheduler splits traffic to the fast subflow (i.e., the subflow providing higher bandwidth). Figures 5.9 and 5.10 show the average fraction of traffic scheduled over the fast subflow and the average throughput measured at the streaming client for the default, DAPS, BLEST, and ECF schedulers As shown in Figure 5.9, ECF allocates traffic to the fast subflow close to the ideal allocation, compared to the other schedulers. By doing this, ECF obtains larger throughputs than other schedulers whenever path asymmetry exists,

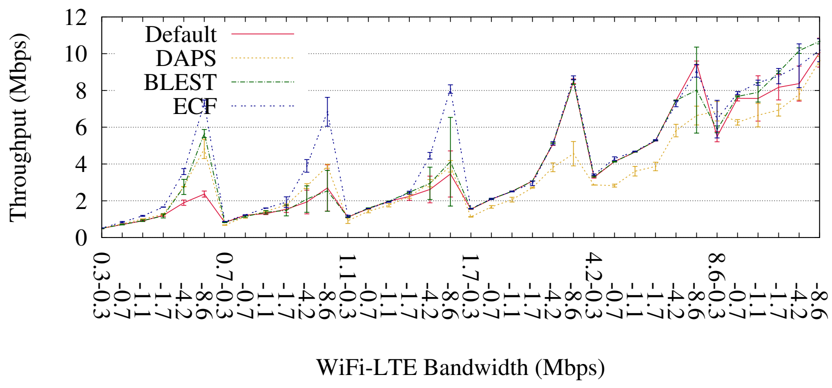
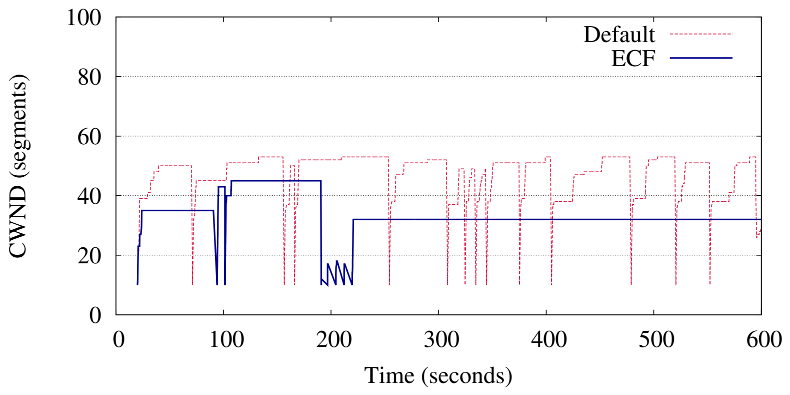


Figure 5.10. Comparison of Throughputs Measured at Streaming Client - Fixed Bandwidth

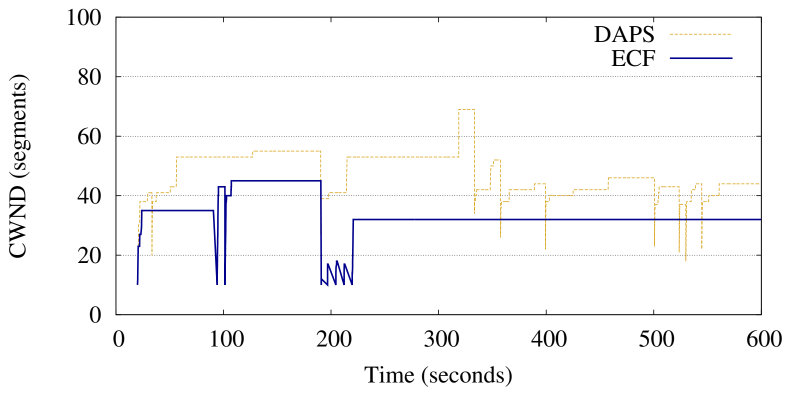
as shown in Figure 5.10. This results in average bit rates close to the ideal average bit rate, as shown in Figure 5.8(b). Note that the fraction of traffic allocated to the fast subflow in the 8.6 Mbps WiFi and 8.6 Mbps LTE pair is larger than the ideal. This is because the 8.6 Mbps WiFi has a smaller RTT (40 ms) than the 8.6 Mbps LTE (105 ms) and transfer sizes (chunk downloads) are not large enough to fully utilize both subflows when bandwidths are large.

5.4.2.3 Congestion Window Behavior

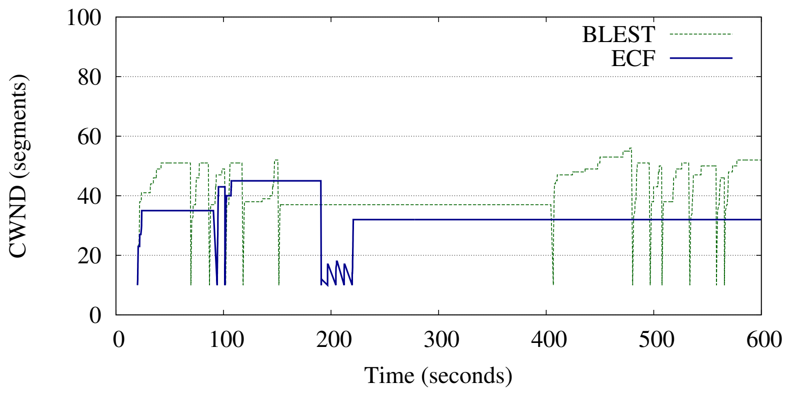
Continuing our investigation, we wish to see the behavior of the congestion window under the different schedulers. Figures 5.11 and 5.12 compare WiFi and LTE CWND behavior of the default and ECF schedulers when WiFi is 0.3 Mbps and LTE is 8.6 Mbps, a streaming case where notable improvement can be seen in Figure 5.8. As shown in Figure 5.11 and 5.12, the default, DAPS, and BLEST schedulers more aggressively utilize the slower, lower-bandwidth WiFi subflow, rather than the faster, higher-bandwidth LTE subflow. In other words, they use the WiFi subflow whenever it is available, but not the faster LTE subflow; we observe that ECF yields the highest utilization of the LTE subflow, followed by BLEST, DAPS, and Default. While the default, DAPS, and BLEST schedulers obtain some more bandwidth from the WiFi



(a) Default



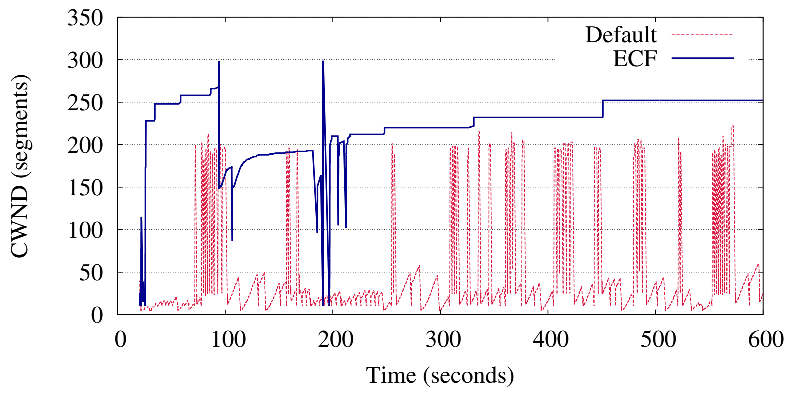
(b) DAPS



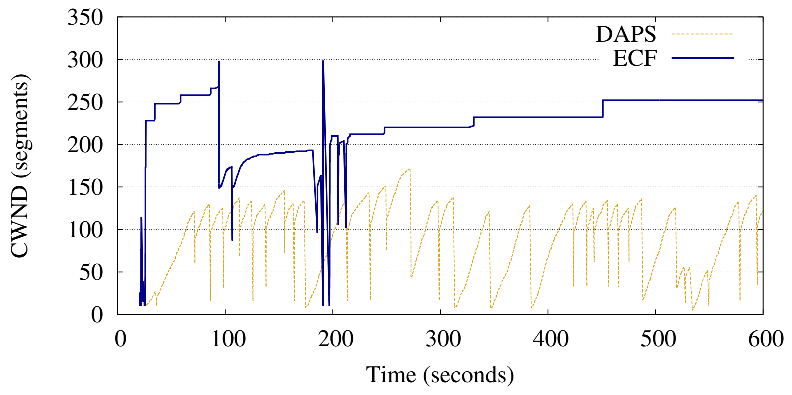
(c) BLEST

Figure 5.11. WiFi CWND Trace Comparison with ECF - 0.3 Mbps WiFi and 8.6 Mbps LTE

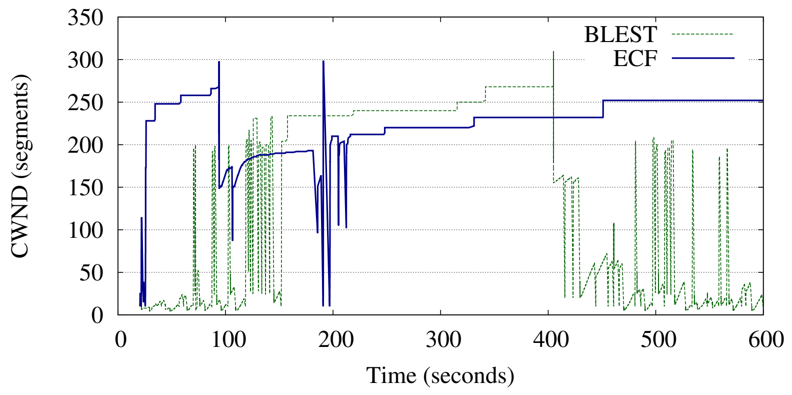
subflow, it makes much less use of LTE and thus causes the LTE subflow to idle, which can cause CWND resets after idle periods longer than the retransmission timeout (see



(a) Default



(b) DAPS



(c) BLEST

Figure 5.12. LTE CWND Trace Comparison with ECF - 0.3 Mbps WiFi and 8.6 Mbps LTE

RFC2861 [27]). Note that the RTT of the LTE subflow (105 ms) is smaller than that of the WiFi subflow (969 ms) and that the idle period is likely to happen at the LTE

Scheduler	Default	DAPS	BLEST	ECF
Average # of Events	2271	362	1742	45

Table 5.3. Number of Events where LTE CWND is set to the IW value - 0.3 Mbps WiFi and 8.6 Mbps LTE

subflow with this bandwidth configuration. Thus, while the WiFi subflow frequently is using a maintained CWND, the LTE subflow unnecessarily starts with an initial CWND of 10 after the idle period in Figure 5.12. This results in under-utilization of the fast (LTE) subflow due to the coupled operation of MPTCP congestion controller; even with the smaller RTT of the LTE subflow, it cannot quickly increase the CWND in Figure 5.12.

In contrast, ECF (solid blue curves) more aggressively uses the LTE subflow and thus suffers fewer restarts from the initial window. Similarly, it makes less use of the WiFi subflow, as indicated by the lower CWND values as compared to the default scheduler. These both results in greater use of the faster subflow and reduce the number of CWND resets after idle periods, thus preserving the feasible values of LTE CWND.

To further study the behavior of the different schedulers in terms of the congestion window, we measure how often the CWND of the LTE subflows is set to the initial window (IW) value, i.e., set back into slow start. Table 5.3 compares the average number of events over the entire video playback, where an event is defined as the LTE CWND being set to the initial window value. Note that these events are caused not only by idle timeouts, but by packet losses as well. As shown in Table 5.3, the default, DAPS, and BLEST schedulers experience high numbers of CWND IW events, while ECF incurs such events only 45 times on average.

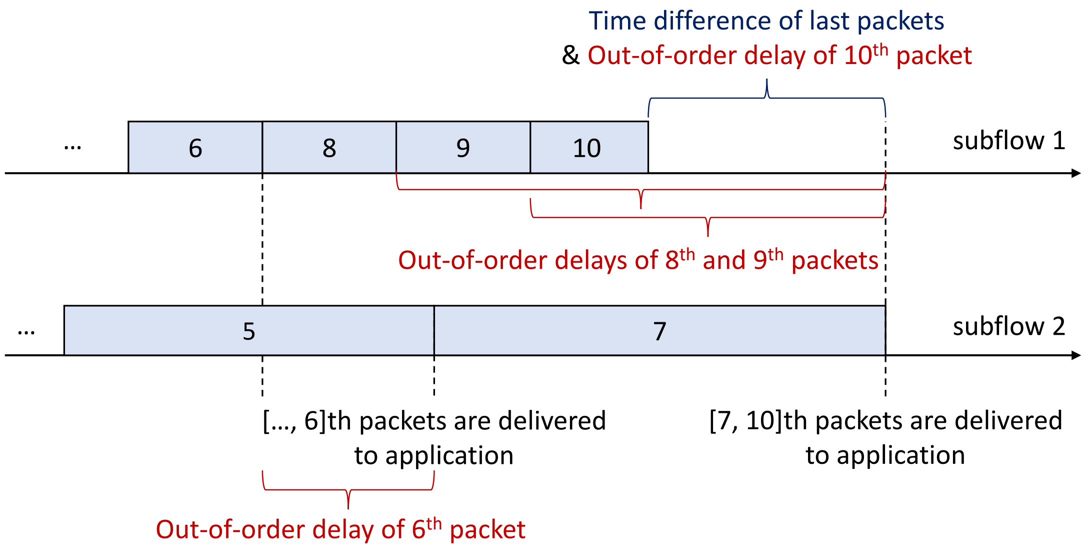

Time difference of last packets
& Out-of-order delay of 10th packet
...
6
8
9
10
subflow 1
Out-of-order delays of 8th and 9th packets
...
5
7
subflow 2
[..., 6]th packets are delivered to application
[7, 10]th packets are delivered to application
Out-of-order delay of 6th packet

Figure 5.13. Out-of-Order Delay

5.4.2.4 Out-of-Order Delay

In the presence of path asymmetry, MPTCP often causes out-of-order delay at the receiver-side, delaying delivery of arrived packets to the application layer. We wish to see how the different schedulers affect this metric. Figure 5.13 shows an example timing diagram of out-of-order delays when a receiver downloads 10 packets: boxes represent download durations of the packet. Since the 6th packet cannot be delivered to the application layer until after the 5th packet arrives over subflow 2, it has to wait in the receive buffer for a while as shown in Figure 5.13.

By the late arrival of the 7th packet over subflow 2, packets 7, 8, 9, and 10 also need to wait in the receive buffer. This wait time that each packet experiences in the receive buffer for in-order delivery to application layer corresponds to out-of-order delay. Since many Internet applications are sensitive to the network quality metric affected by out-of-order delays, most notably real-time streaming, it is important for MPTCP path schedulers to minimize out-of-order delays.

Figure 5.14 presents the CDF and CCDF of the out-of-order delay that individual packets experience with the default scheduler. As shown in Figure 5.14, the default

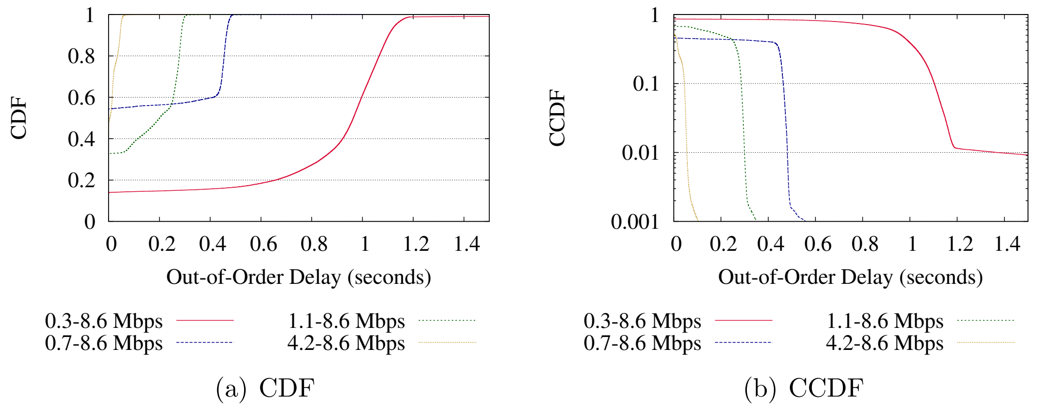
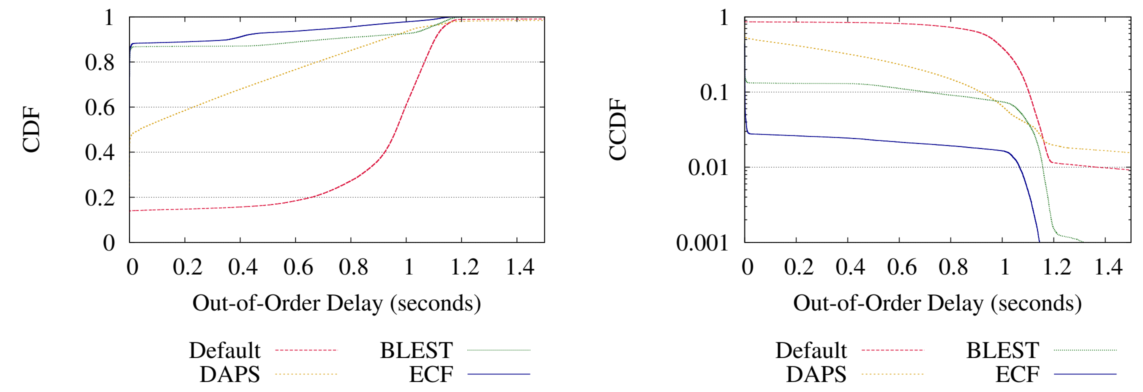


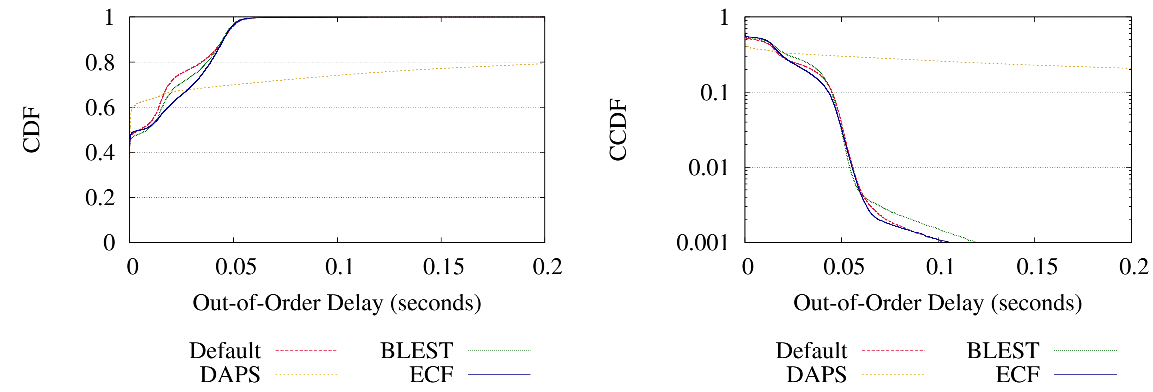
Figure 5.14. Out-of-Order Delay (Default Scheduler)

scheduler yields longer out-of-order delays as the path asymmetry becomes larger. In addition, we observe that out-of-order delays are strongly related to the time difference between the last packets (compare Figure 5.15 with Figure 5.3(b)). In other words, the larger time difference of last packets is likely to be triggered by larger out-of-delay at the end of download completion.

Figure 5.15 compares the CDF and CCDF of out-of-order delay of each scheduler under two bandwidth configurations. As shown in Figure 5.15(a), DAPS, BLEST, and ECF yield smaller out-of-order delays than the default scheduler in 0.3 Mbps WiFi and 8.6 Mbps LTE pair. Of these all the schedulers, ECF performs the best, i.e., under ECF, almost 98% of packets experience out-of-order delays smaller than one second (the out-of-delays of only 2% of packets are longer than one second), while with the default scheduler more than 40% of packets suffer from out-of-order delays larger than one second and around 12% of packets do with DAPS and BLEST. In Figure 5.15(b), we observe that out-of-order delay behavior becomes similar across the schedulers (except for DAPS) as path asymmetry becomes smaller; the schedulers mostly yield out-of-order delays of 0.05 seconds (again, except for DAPS). On the other hand, under DAPS, more than 30% of packets are delivered to the application



(a) 0.3 Mbps WiFi and 8.6 Mbps LTE



(b) 4.2 Mbps WiFi and 8.6 Mbps LTE

Figure 5.15. Comparison of Out-of-Order Delay (Streaming - CDF & CCDF)

layer with a delay larger than 0.06 sec, which is even larger than that of the default scheduler.

5.4.3 Video Streaming with Bandwidth Changes

Next, we examine how ECF responds to changes in network bandwidth for video streaming workloads. Here, WiFi and LTE bandwidths change randomly at exponentially distributed intervals of time with an average of 40 seconds. The bandwidths are selected from the set $\{0.3, 1.1, 1.7, 4.2, 8.6\}$ Mbps, chosen uniformly at random. Throughputs are measured at the streaming client using ten scenarios with different random seeds. For each scenario, we collect traces from three runs.

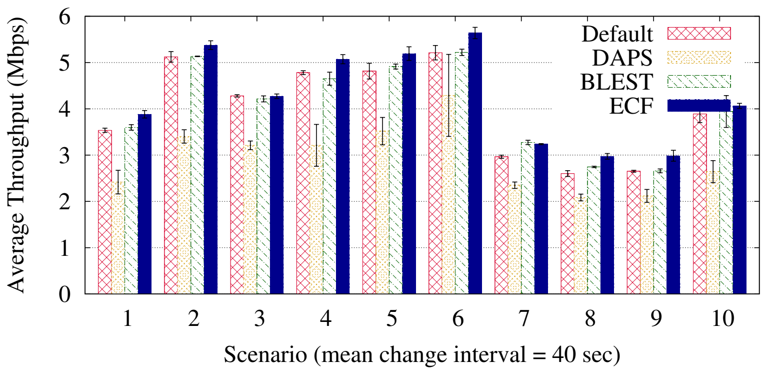


Figure 5.16. Throughputs Measured at Streaming Client - Random Bandwidth Changes

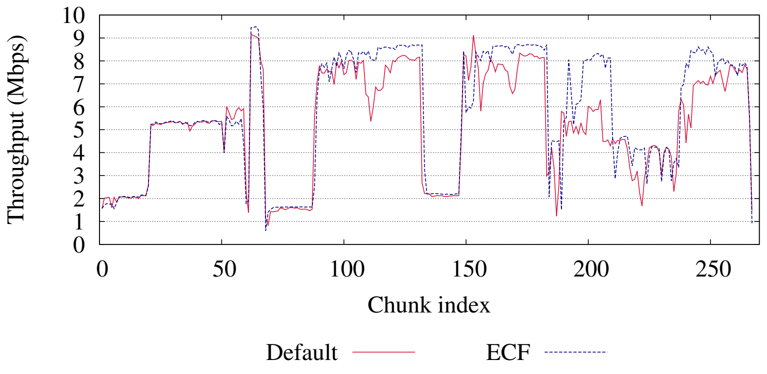


Figure 5.17. Example Throughput Trace - Streaming with Random Bandwidth Changes

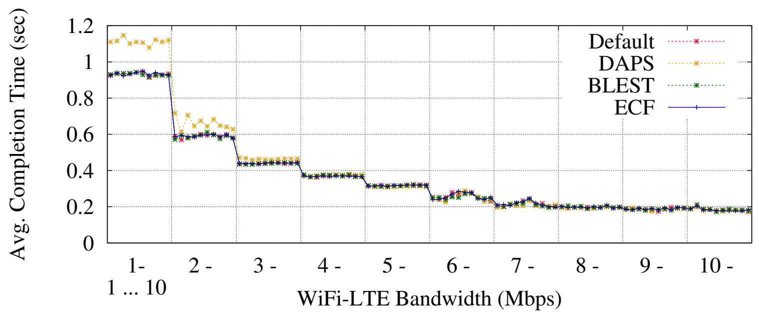
Figure 5.16 compares the average throughputs using the default, ECF, and BLEST schedulers for each random scenario. As can be seen, ECF outperforms the other schedulers in terms of average throughput, producing the highest average streaming bit rate. Average bit rate exhibits similar behavior. Figure 5.17 presents the measured throughput for each chunk download for an individual random scenario (number 6 in Figure 5.16). We observe that ECF yields similar or larger download throughput than the default scheduler for any streaming chunk download. In particular, ECF makes more efficient use of the faster subflow in the presence of path asymmetry and otherwise yields at least similar performance as the default scheduler.

5.4.4 Simple Web Downloads

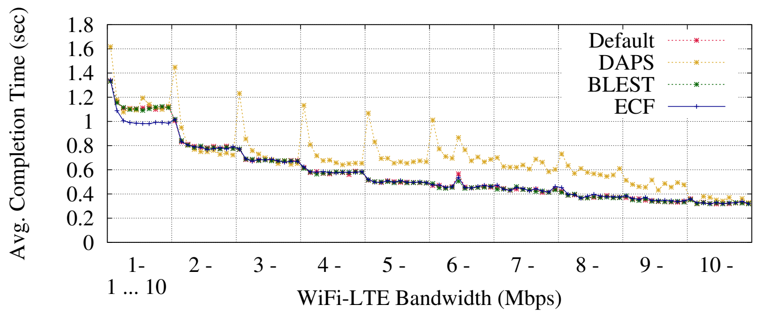
Next, we examine the performance of the four schedulers using wget. The purpose of these experiments is to show that, for simple downloads, ECF improves performance in the presence of path asymmetries, while not degrading performance when paths are similar. We measure wget download completion times for several file sizes (64 KB to 2 MB) for the WiFi and LTE bandwidths of $1, 2, ..., 10$ Mbps as in Section 5.1.2. Since in these experiments, MPTCP transfers a single object in a shorter time than the streaming experiments, we expect that the performance difference across the schedulers to be less significant; an idle period of the fast subflow only appears once and a CWND reset after an idle period never occurs.

Figure 5.18 presents the completion times when downloading a range of selected file sizes, for all schedulers, averaged over thirty runs. First, ECF does no worse statistically than the default scheduler, and occasionally does better when paths are asymmetric and transfers large. Second, as expected, Figure 5.18 shows in most cases no notable differences between the schedulers except for DAPS. Recall that WiFi is the primary subflow. MPTCP rarely utilizes a secondary subflow (LTE in this case) for small transfers [7]. Therefore, unless the primary path (WiFi) is extremely slow, path schedulers do not affect performance for small downloads such as 128 KB downloads. However, DAPS sometimes yields larger average completion times, e.g., for the 128 KB download case where WiFi is 1 Mbps and LTE ranges from [1, 10] Mbps. We attribute this to the strong dependency of DAPS on the RTT ratio; an incorrect estimate of the LTE RTT results in unnecessary trials to inject traffic into the slow LTE subflow.

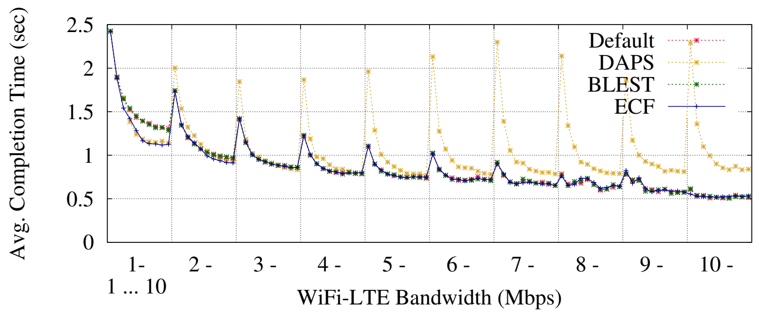
To compare performance between the default and ECF scheduler, Figure 5.19 shows download completion times of the ECF scheduler normalized relative to that of the default scheduler. To plot this figure, we set the normalized value to one if the download time difference between the ECF and default scheduler is in the



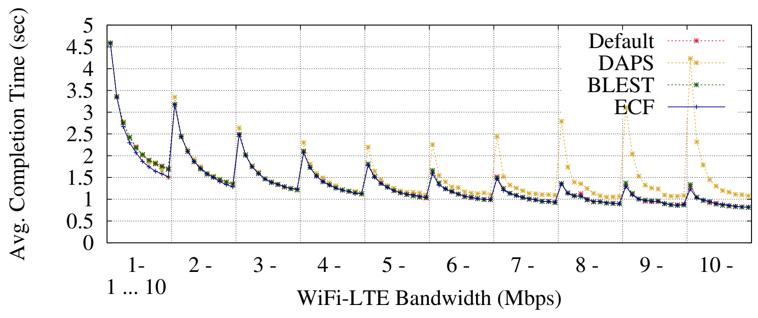
(a) 128KB



(b) 256KB



(c) 512KB



(d) 1M

Figure 5.18. Average Download Completion Time - Various File Sizes (lower is better)

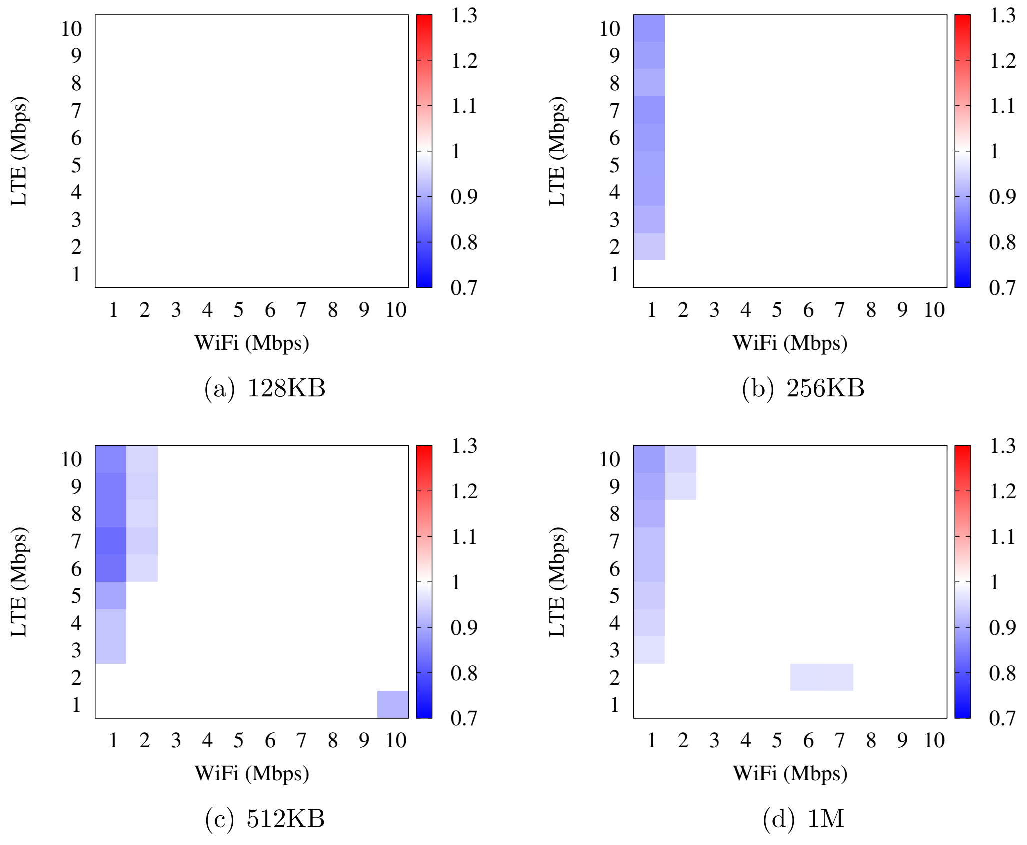


Figure 5.19. ECF Average Download Completion Time Ratio Normalized by Default

range of their standard deviation. Otherwise, the ratio is defined as the ratio of the averages. Thus, the value of one in Figure 5.19 means that both of the default and ECF schedulers yield similar performance, and smaller than one (more blue) means that the ECF scheduler takes shorter time than the default scheduler.

As shown in Figure 5.19(a), for small transfers (128 KB), both the default and ECF schedulers yield similar completion times. We observe notable performance difference for downloads of 256 KB and greater. Figures 5.19(b)-(c) shows that ECF yields up to 20% smaller download times than the default scheduler in the presence of path asymmetries when downloading files larger than or equal to 256 KB. Note that the improvement by ECF scheduler becomes smaller as the transfer size increases. This is because the wget experiments examine the performance of single object download

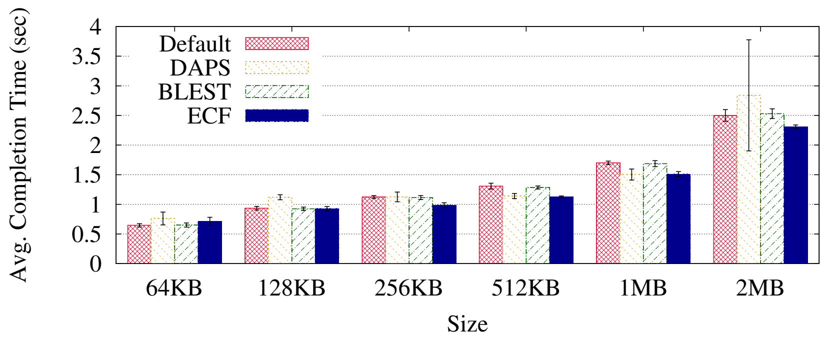


Figure 5.20. Average Download Completion Time - 1.0 Mbps WiFi and 10.0 Mbps LTE (lower is better)

over an MPTCP connection, in which an idle period of the fast subflow only appears once and total transfer time becomes comparatively longer than such a idle period with a larger transfer.

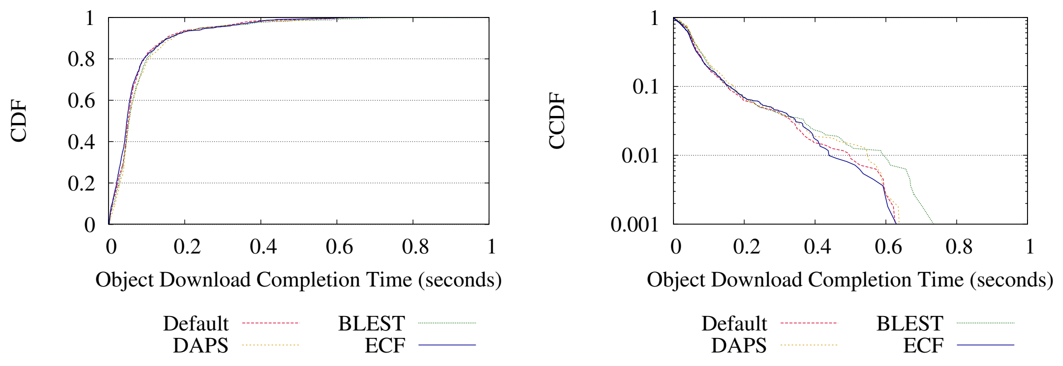
Figure 5.20 shows average download completion time (from thirty runs) as a function of the download size when WiFi is 1 Mbps and LTE is 10 Mbps, an extreme case where incorrect scheduling decisions affect MPTCP performance. ECF appears to outperform all other schedulers except for small transfer cases such as 64 KB and 128 KB. In the case of 64 KB downloads, the average completion time of ECF appears larger than that of the default scheduler, however the difference is not statistically significant considering the extremely short transfer time for 64KB download and its standard deviation As we see in Figure 5.20, BLEST does not achieve any performance improvement compared to the default scheduler while DAPS enhances performance only for downloads of 512 KB and 1 MB (note that DAPS yields worse performance for such transfer sizes under other bandwidth pairs than the default scheduler in Figure 5.18). In particular, BLEST yields almost the same average download completion time as the default scheduler and DAPS does not exhibit consistent performance (better or worse with large standard deviations e.g., the case of 2 MB). In contrast, ECF exhibits a smaller average download time (up to 20%) than the default after the download size becomes larger than 256 KB.

5.4.5 Web Browsing

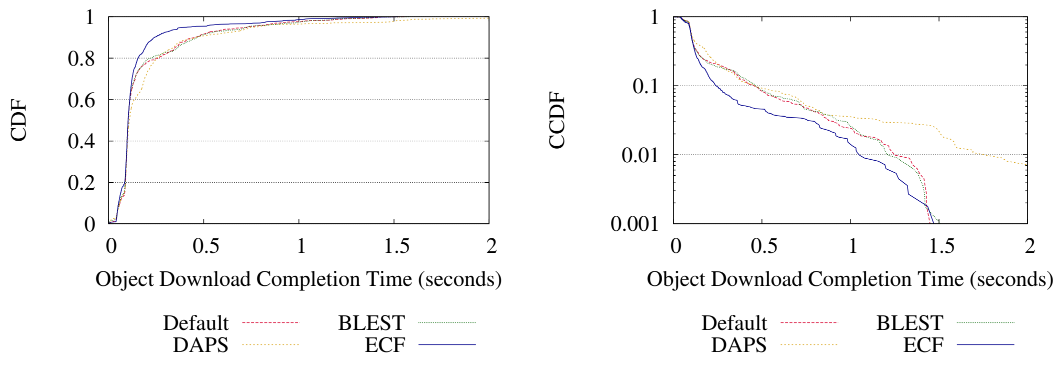
We now examine whether ECF improves the performance of a common application: Web browsing. We deploy a copy of CNN's home page (as of 9/11/2014), which consists of 107 Web objects, into our MPTCP server. Web-browsing activity is similar with a series of wget downloads over one connection, therefore, such consecutive downloads over one MPTCP connection can be affected more by idle periods of the fast subflow and CWND resets compared to a single object download using wget. To see how each scheduler affects Web object download performance, we investigate the distribution of object download completion times while regulating the WiFi and LTE bandwidths between [1,10] Mbps as is in Section 5.4.4. In this experiment, the Android web browser establishes six parallel (MP)TCP connections to the server (12 subflows for MPTCP), with HTTP persistent connections. Note we collect traces from 10 runs.

Figure 5.21 compares the CDFs and CCDFs of individual object download completion times of each scheduler with three bandwidth configurations. In Figure 5.21(a), with the symmetric regulated bandwidth (5.0 Mbps), we observe that all the schedulers yields almost same download completion time: 98% of object downloads are completed in similar time for all schedulers. As shown in Figure 5.21(b), with 1.0 Mbps WiFi and 5.0 Mbps LTE pair, ECF completes 99% of object downloads earlier (at least in similar time) than the other schedulers. In this bandwidth pair, BLEST yields almost the same performance as the default scheduler and DAPS does not achieve any performance gain, as with the streaming and simple Web download experiments. In Figure 5.21(c), we observe that as path asymmetry becomes larger, ECF more explicitly exhibits smaller object download completion times than the other schedulers, while DAPS and BLEST do not outperform the default scheduler.

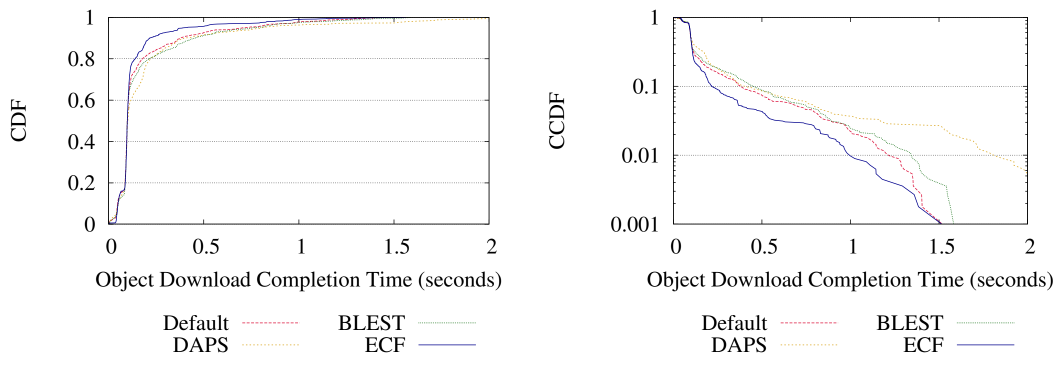
Figure 5.22 presents the CDFs of the out-of-order delay that individual packets experiences while the browser downloads web objects under the three bandwidth



(a) 5.0 Mbps WiFi and 5.0 Mbps LTE



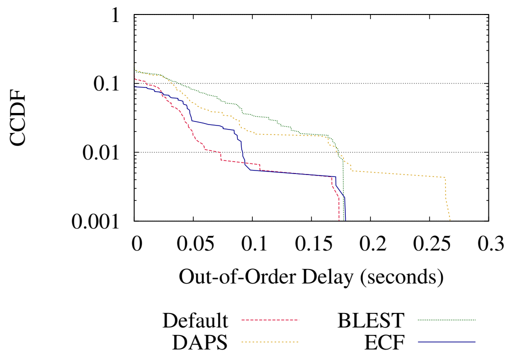
(b) 1.0 Mbps WiFi and 5.0 Mbps LTE



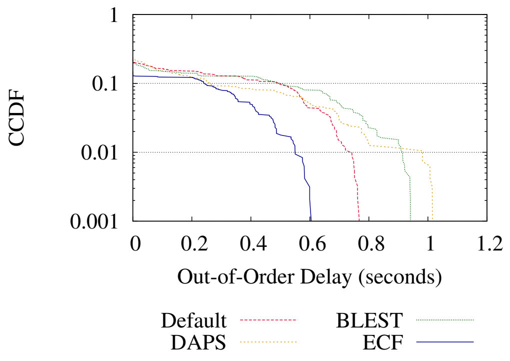
(c) 1.0 Mbps WiFi and 10.0 Mbps LTE

Figure 5.21. Web Object Download Completion Time (CDF & CCDF)

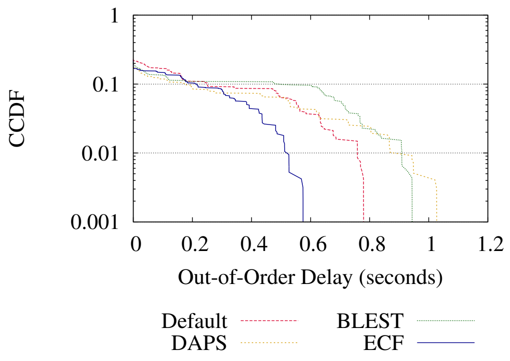
configurations. As with Figure 5.15 for the streaming cases, we observe that ECF successfully mitigates out-of-order delay in Web browsing activities as path becomes asymmetric.



(a) 5.0 Mbps WiFi and 5.0 Mbps LTE



(b) 1.0 Mbps WiFi and 5.0 Mbps LTE



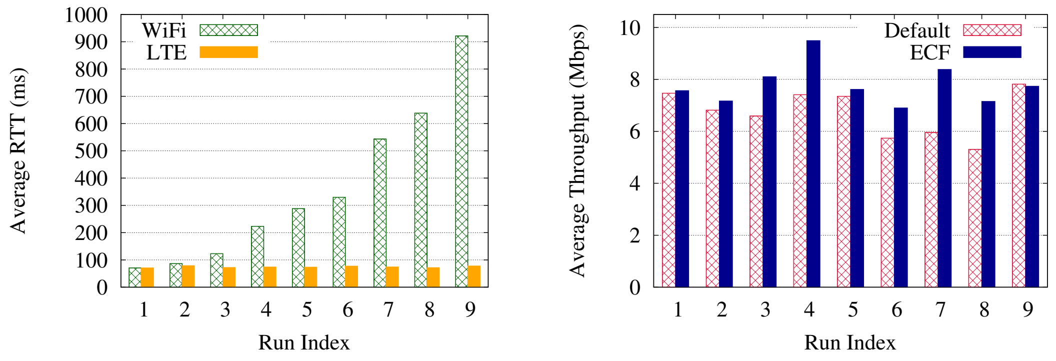
(c) 1.0 Mbps WiFi and 10.0 Mbps LTE

Figure 5.22. Comparison of Out-of-Order Delay (Web Browsing)

5.5 Evaluation in the Wild

5.5.1 Experimental Setup

We next examine whether the ECF scheduler provides better performance than the default scheduler in more realistic environments. We limit our comparison of ECF with the default scheduler since the other schedulers do not exhibit consistent improvement over the default scheduler in the previous experiments. To this end, we deploy an MPTCP enabled server in Washington D.C. using IBM SoftLayer Cloud, which uses the same configurations to the server for controlled in-lab experiments. The mobile device communicates with the server over the Internet using a WiFi access point (town public WiFi in Amherst, MA) and an LTE cellular interface from



(a) RTT (b) Throughput

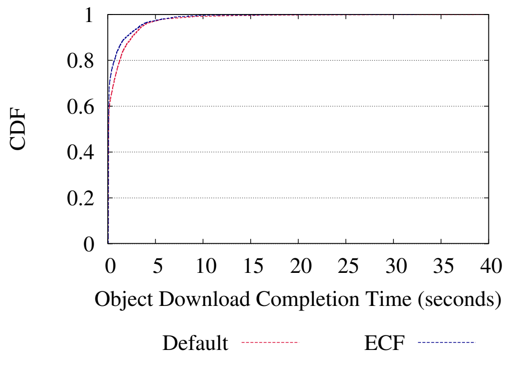
Figure 5.23. Streaming Experiments in Wild

AT&T. Note that in these experiments, the device uses each network as it is without any additional bandwidth regulation.

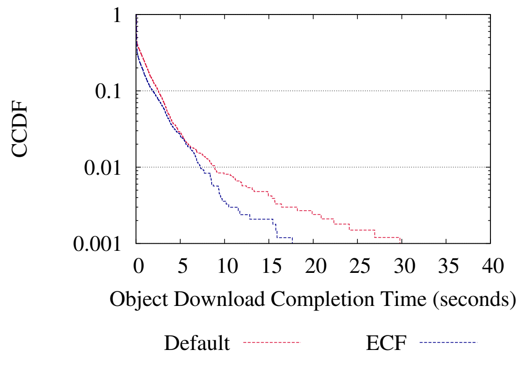
5.5.2 Video Streaming in the Wild

We first explore the streaming performance over MPTCP using the default and ECF schedulers in the wild configuration. We perform nine runs over two days using our streaming workload on the WDC server. Figure 5.23(a) shows the average measured RTT for each run. Note that results are sorted by WiFi average RTT. We observe that LTE has a consistent average RTT (around 70 ms). As shown in Figure 5.23(a), runs 1 and 2 have similar WiFi and LTE RTTs, that is, both paths are symmetric in terms of RTT, resulting in similar performance between the default and ECF schedulers. Since there are significant differences between WiFi and LTE RTTs in runs 4-9, the default scheduler is likely to experience throughput degradation whereas ECF is not.

Figure 5.23(b) presents the average throughputs obtained by the streaming client using the default and ECF schedulers. As expected, both schedulers yield similar throughputs in runs 1 and 2. In later runs, the differences in RTT between WiFi and LTE become larger, resulting in larger average throughputs for ECF than for the



(a) CDF



(b) CCDF

Figure 5.24. Web Object Download Completion Time - Experiments in Wild

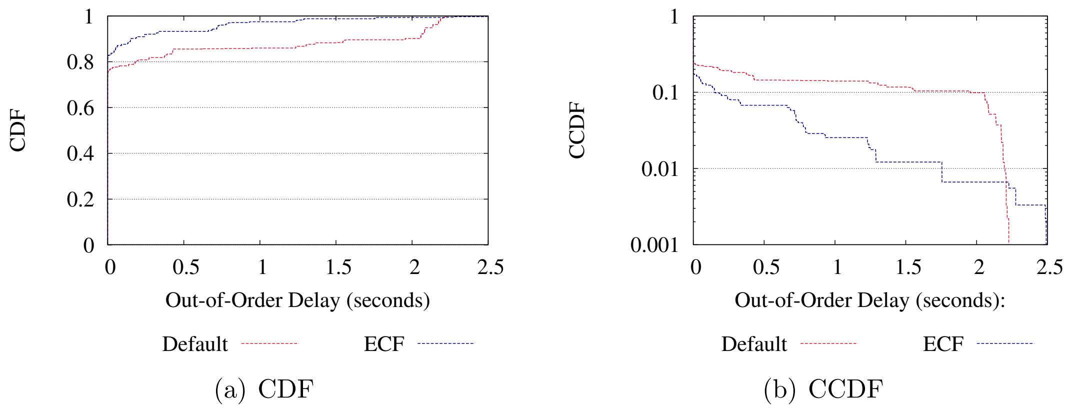
default scheduler. However, both schedulers again obtain similar average throughputs in run 9. Note that the WiFi subflow is the primary subflow and in run 9, the WiFi average RTT is close to one second, which is more than ten times larger than the LTE RTT. In this case, both schedulers only use the WiFi subflow for the first few packets at the beginning of the HTTP GET response, resulting in similar performance. On average, the ECF throughput is 7.79 Mbps while the default scheduler 6.72 Mbps, an improvement of 16%.

5.5.3 Web browsing in the Wild

Next, we investigate the distribution of the object download completion times when the device retrieves a copy of CNN's home page at the WDC server, measured over thirty runs. Figure 5.24 compares the CDFs and CCDFs of the individual object download completion times of the default and ECF schedulers. The average statistics are listed in Table 5.4. As shown in Figure 5.24, ECF yields smaller object download completion times than the default scheduler. On average, ECF completes the object downloads in 0.65 seconds, while the default scheduler requires 0.88 seconds, an improvement of 26%. In addition, while ECF completes 99.9% of object downloads in around 17 seconds, the default scheduler does in 30 seconds.

	Download Completion Time (sec)	Out of Order Delay (sec)
Default	0.882	0.297
ECF	0.650	0.087
ECF Improvement	26% shorter	71% shorter

Table 5.4. Average Statistics of Web Browsing in the Wild



(a) CDF (b) CCDF

Figure 5.25. Comparison of Out-of-Order Delay - Experiments in Wild

Figure 5.25 presents the CDFs and CCDFs of the out-of-order delay that individual packets experience. As shown in Figure 5.25, 99% of packets downloaded using ECF experience smaller out-of-order delays than with the default scheduler. ECF yields an average out-of-order delay of 0.087 seconds, while the default scheduler yields an average of 0.297 seconds, an improvement of 71%. Only 0.2% of packets downloaded using ECF exhibit slightly larger out-of-order delays than the largest one using the default scheduler. We found that 0.2% is from twelve instances out of approximately 27000 data points; these twelve packets suffer out-of-order delays of approximately 2.5 seconds.

5.6 Related Work

Although the design of the MPTCP path scheduler significantly impacts performance and quality of experience, there have not been many practical studies of improved MPTCP path schedulers that have been implemented and evaluated experimentally.

Raiciu et al. [63] point out that path heterogeneity can result in performance degradation due to head of line blocking or limited receive window size due to reorderings. To resolve these problems, they propose opportunistic retransmission and penalization mechanisms, which are included in the Linux MPTCP Kernel implementation. These mechanisms have been evaluated in more detail in [53, 54].

Kuhn et al. [39] propose delay-aware packet scheduling for MPTCP. This approach considers large path asymmetries in delay and stable CWND, but does not take advantage of the send buffer. In addition, it is evaluated only by ns2 simulations.

Ferlin et al. [18] propose a scheduler to prevent fast subflow blocking due to path asymmetry. Their scheduler waits for a fast subflow if during the RTT of the slow path, the fast subflow can transfer more packets than the available space in the connection-level send window. However, it does not consider idle fast subflow due to nothing to send.

Yang et al. [78] suggest a scheduler that distributes traffic proportional to the estimated path capacity. However, they only consider scenarios with very large transfers in a network with a small amount of buffering.

Corbillion [10] propose a scheduler to improve streaming video performance over MPTCP. They do not implement their approach and evaluate it only via simulation. In addition, their solution requires modifying the video sending application to integrate it with the MPTCP scheduler, whereas our approach is application-independent.

Nikravesh et al. [50] present a measurement study of MPTCP in the wild, and propose MPFLEX, an architecture for supporting multipath over mobile networks. However, MPFLEX is not compatible with MPTCP and requires modifications to both the client and server.

Han et al. [26] present MP-DASH, a framework for scheduling streaming video traffic over MPTCP. They show that by exploiting knowledge of video streaming, traffic can be scheduled so as to significantly reduce cellular usage and power consumption with negligible degradation of QoE. Their approach, however, requires modifications to both the client and server, and is focused solely on video traffic. ECF, in contrast, is a server-side only modification, improving deployability, and works transparently for multiple workloads, not just streaming video.

5.7 Conclusion

In this chapter, we demonstrated that the MPTCP default path scheduler degrades performance in the presence of path asymmetry, even with the reinjection and penalization mechanisms. We identified the root cause of the problem: paths are under-utilized due to idle periods and CWND resets caused by idle periods. To resolve this problem, we proposed a novel MPTCP path scheduler (ECF) that predicts and avoids potential idle periods of fast paths given congestion window, round trip time, and the amount of remaining transfer size. Our experimental results show that ECF outperforms the existing schedulers in several scenarios with simple Web downloads and with video streaming.

CHAPTER 6
SUMMARY AND FUTURE DIRECTIONS

6.1 Thesis Summary

This thesis studied the MPTCP deployment issues in mobile devices. We first explored the performance degradation problem of MPTCP under mobile environment with dynamic connectivity. We then investigate the MPTCP energy consumption behavior using off the shelf mobile devices. Based on the MPTCP energy model, we illustrate the tradeoffs in MPTCP between link bandwidth, transfer size, network performance and energy consumption. Finally, we study the impact of path asymmetry on MPTCP performance using several types of Internet traffic load such as video streaming and Web browsing.

Chapter 2 presented a cross-layer path management scheme, called MPTCP-MA, to mitigate the performance degradation caused by under-utilization of recovered paths in mobile environment. MPTCP-MA exploits information from not only transport layer but also medium-access-control and physical layer to estimate path status and correspondingly control path usage. MPTCP-MA has been implemented in an actual mobile device and evaluated with a mobile scenario. The experimental results proved that MPTCP-MA successfully controls path usage to avoid excessive loss related procedures at the transport layer so that it can more quickly utilize recovered paths, achieving higher throughput than standard MPTCP.

In Chapter 3, we conducted the measurement study on MPTCP energy consumption behavior. The measurement has been done using off the shelf mobile devices under various network conditions. Based on the collected measurement results, we developed the regression model to estimate MPTCP energy consumption given path status and transfer size. The model was validated against the actual measured energy consumption of the mobile devices and the results demonstrated that the model correctly estimates MPTCP energy consumption with small root mean square errors.

In Chapter 4, we further studied how to enhance MPTCP to reduce energy consumption while preserving the gains from using multiple paths such as greater throughput and robustness. Based on the energy model in Chapter 3, we explored conditions under which MPTCP becomes more energy-efficient than either standard TCP or MPTCP. This finding was the basis of the design of energy aware MPTCP, called eMPTCP; eMPTCP seeks to choose path usage that provides better energy efficiency given network conditions and transfer sizes. We evaluated eMPTCP accross a range of scenarios such as controlled in-lab and in-wild experiments. The experimental results exhibited that eMPTCP appropriately controls path usage for better energy efficiency; it achieves energy gain by suspending use of inefficient paths in terms of energy.

In Chapter 5, we investigated the MPTCP performance issues in the presence of path asymmetries. We figured out that the default path scheduler causes idle periods with higher bandwidth paths, resulting in lower throughput than actual available aggregate bandwidth. We developed and implemented the new MPTCP path scheduler, called ECF (Earliest Completion First), which predicts and avoids potential idle periods of fast paths caused by use of slow paths given congestion window, round trip time, and the amount of remaining transfer size. ECF was evaluated with several traffic load such as video streaming, simple Web object downloads, and Web browsing. The experimental results showed that ECF mitigates the under-utilization of fast subflow due to their undesired idle periods while reducing out-of-order delay from transport to application layer.

6.2 Future Work

Video streaming such as YouTube and Netflix is one of the most popular applications in mobile devices, of which traffic accounts for around 40% of the peak mobile downstream bytes in North America [68]. Such video streaming services operate over HTTP and TCP using dynamic bit rate adaptation, Dynamic Adaptive Streaming over HTTP (DASH), to provide high quality user experience in terms of streaming quality and latency. Despite several state-of-art strategies to ensure good quality of user experience in video streaming over standard TCP, relatively little work has been done to study how video streaming behaves over MPTCP.

I will seek to improve on the state-of-art streaming applications working together with MPTCP, incorporating the lessons from my thesis work. To this end, I will first model the behavior of DASH applications according to streaming setup and network conditions, such as traffic on-off cycle, that affects MPTCP performance in terms of path utilization and energy efficiency. As a final goal of this study, based on the analytical study on DASH behavior, I will develop and implement a framework for mobile video streaming using MPTCP (mDASH), aimed at reducing energy consumption as well as cellular data usage while preserving video streaming quality: mDASH is an enhanced streaming scheme based on DASH, which collaborates with MPTCP in order to provide improved user experience such as better streaming quality and less energy consumption.

BIBLIOGRAPHY

[1] Ahrenholz, J. Comparison of CORE network emulation platforms. In *Proc. of MILCOM* (2010), pp. 166–171.

[2] Astely, D., Dahlman, E., Furuskar, A., Jading, Y., Lindstrom, M., and Parkvall, S. LTE: The evolution of mobile broadband. *Communications Magazine, IEEE 47*, 4 (2009), 44–51.

[3] Balasubramanian, Niranjan, Balasubramanian, Aruna, and Venkataramani, Arun. Energy consumption in mobile phones: A measurement study and implications for network applications. In *Proc. of ACM IMC* (2009), pp. 280–293.

[4] Bui, Duc Hoang, Lee, Kilho, Oh, Sangeun, Shin, Insik, Shin, Hyojeong, Woo, Honguk, and Ban, Daehyun. Greenbag: Energy-efficient bandwidth aggregation for real-time streaming in heterogeneous mobile wireless networks. In *Proc. of IEEE RTSS* (2013), pp. 57–67.

[5] Carroll, Aaron, and Heiser, Gernot. An analysis of power consumption in a smartphone. In *Proc. of USENIX Annual Technical Conference (ATC)* (2010), pp. 21–21.

[6] Carroll, Aaron, and Heiser, Gernot. The systems hacker's guide to the galaxy: Energy usage in a modern smartphone. In *Proc. of APSys* (2013), pp. 5:1–5:7.

[7] Chen, Yung-Chih, Lim, Yeon-Sup, Gibbens, Richard J, Nahum, Erich, Khalili, Ramin, and Towsley, Don. A measurement-based study of multipath TCP performance in wireless networks. In *Proc. of ACM IMC* (2013), pp. 455–468.

[8] Choi, Jaehyuk, Na, Jongkeun, Lim, Yeon-Sup, Park, Kihong, and Kim, Chong-Kwon. Collision-aware design of rate adaptation for multi-rate 802.11 WLANs. *IEEE Journal on Selected Areas in Communications 26*, 8 (October 2008), 1366 –1375.

[9] Cipriano, A. M., Gagneur, P., Vivier, G., and Sezginer, S. Overview of ARQ and HARQ in beyond 3G systems. In *Proc. of IEEE PIMRC Workshops* (2010), pp. 424–429.

[10] Corbillon, Xavier, Aparicio-Pardo, Ramon, Kuhn, Nicolas, Texier, Géraldine, and Simon, Gwendal. Cross-layer scheduler for video streaming over MPTCP. In *Proc. of ACM MMSys* (2016), pp. 7:1–7:12.

[11] Croitoru, Andrei, Niculescu, Dragos, and Raiciu, Costin. Towards WiFi mobility without fast handover. In *Proc. of USENIX NSDI* (2015), pp. 219–234.

[12] CynogenMod. D2att info, 2012. `http://wiki.cyanogenmod.org/index.php?title=D2att_Info` [Online; The source code of Android Platform is retrieved from GIT at 10-Dec-2012].

[13] De Couto, Douglas S. J., Aguayo, Daniel, Bicket, John, and Morris, Robert. A high-throughput path metric for multi-hop wireless routing. In *Proc. of ACM MobiCom* (2003), pp. 134–146.

[14] Deng, Shuo, Netravali, Ravi, Sivaraman, Anirudh, and Balakrishnan, Hari. WiFi, LTE, or both? Measuring multi-homed wireless Internet performance. In *Proc. of ACM IMC* (2014), pp. 181–194.

[15] Detal, Gregory. MPTCP-enabled kernel for the Nexus 5. `https://github.com/gdetal/mptcp_nexus5`.

[16] Ding, A.Y., Han, Bo, Xiao, Yu, Hui, Pan, Srinivasan, A., Kojo, M., and Tarkoma, S. Enabling energy-aware collaborative mobile data offloading for smartphones. In *Proc. of IEEE SECON* (2013), pp. 487–495.

[17] Ding, Ning, Wagner, Daniel, Chen, Xiaomeng, Hu, Y Charlie, and Rice, Andrew. Characterizing and modeling the impact of wireless signal strength on smartphone battery drain. In *Proc. of ACM SIGMETRICS* (2013), pp. 29–40.

[18] Ferlin, Simone, Alay, Ozg, Mehani, Olivier, and Boreli, Roksana. BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks. In *Proc. of IFIP Networking* (2016), pp. 1222–1227.

[19] Fielding, R., and Reschke, J. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. *IETF RFC 7230* (2014).

[20] Ford, A., Raiciu, C., Handley, M., Barre, S., and Iyengar, J. Architectural guidelines for multipath TCP development. *IETF RFC 6182 (Informational)* (2011).

[21] Ford, Alan, Raiciu, Costin, Handley, Mark, and Bonaventure, Olivier. TCP extensions for multipath operation with multiple addresses. *IETF RFC 6824* (2013).

[22] Garcia-Saavedra, Andres, Serrano, Pablo, Banchs, Albert, and Bianchi, Giuseppe. Energy consumption anatomy of 802.11 devices and its implication on modeling and design. In *Proc. of ACM CoNEXT* (2012), pp. 169–180.

[23] Goff, T., Moronski, J., Phatak, D.S., and Gupta, V. Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments. In *Proc. of IEEE INFOCOM* (2000), pp. 1537–1545 vol.3.

[24] Google. Exoplayer. `http://google.github.io/ExoPlayer/`.

[25] Halperin, Daniel, Greenstein, Ben, Sheth, Anmol, and Wetherall, David. Demystifying 802.11n power consumption. In *Proceedings of the 2010 workshop on Power aware computing and systems (HotPower)* (2010), USENIX Association, p. 1.

[26] Han, Bo, Qian, Feng, Ji, Lusheng, and Gopalakrishnan, Vijay. MP-DASH: Adaptive video streaming over preference-aware multipath. In *Proc. of ACM CoNEXT* (2016), pp. 129–143.

[27] Handley, M., Padhye, J., and Floyd, S. TCP congestion window validation. *IETF RFC 2861* (2000).

[28] He, Qi, Dovrolis, Constantine, and Ammar, Mostafa. On the predictability of large transfer TCP throughput. In *Proc. of ACM SIGCOMM* (2005), pp. 145–156.

[29] Holland, Gavin, and Vaidya, Nitin. Analysis of TCP performance over mobile ad hoc networks. In *Proc. of ACM MobiCom* (1999), pp. 219–230.

[30] Huang, Junxian, Feng, Qiang, Gerber, Alexandre, Mao, Z. Morley, Sen, Subhabrata, and Spatscheck, Oliver. A close examination of performance and power characteristics of 4G LTE networks. In *Proc. of ACM MobiSys* (2012), pp. 225–238.

[31] Huang, Te-Yuan, Johari, Ramesh, McKeown, Nick, Trunnell, Matthew, and Watson, Mark. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proc. of ACM SIGCOMM* (2014), pp. 187–198.

[32] IEEE. Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications.

[33] JackFrags. 4k gaming montage. `http://4ksamples.com/4k-gaming-montage/`.

[34] Jang, Ki-Young, Hao, Shuai, Sheth, Anmol, and Govindan, Ramesh. Snooze: Energy management in 802.11n WLANs. In *Proc. of ACM CoNEXT* (2011), pp. 12:1–12:12.

[35] Judd, Glenn, and Steenkiste, Peter. Understanding link-level 802.11 behavior: Replacing convention with measurement. In *Proc. of ICST WICON* (2007), pp. 19:1–19:10.

[36] Kelly, Frank, and Voice, Thomas. Stability of end-to-end algorithms for joint routing and rate control. *SIGCOMM Comput. Commun. Rev. 35*, 2 (Apr. 2005), 5–12.

[37] Khalili, Ramin, Gast, Nicolas, Popovic, Miroslav, Upadhyay, Utkarsh, and Le Boudec, Jean-Yves. MPTCP is not pareto-optimal: Performance issues and a possible solution. In *Proc. of ACM CoNEXT* (2012), pp. 1–12.

[38] Klemm, Fabius, Ye, Zhenqiang, Krishnamurthy, Srikanth V., and Tripathi, Satish K. Improving TCP performance in ad hoc networks using signal strength based link management. *Ad Hoc Networking 3*, 2 (Mar. 2005), 175–191.

[39] Kuhn, N., Lochin, E., Mifdaoui, A., Sarwar, G., Mehani, O., and Boreli, R. DAPS: Intelligent delay-aware packet scheduling for multipath transport. In *Proc. of IEEE ICC* (2014), pp. 1222–1227.

[40] Li, Chengzhou, and Papavassiliou, S. The link signal strength agent (LSSA) protocol for TCP implementation in wireless mobile ad hoc networks. In *Proc. of IEEE VTC* (2001), pp. 2528–2532 vol.4.

[41] Li, Chi-Yu, Peng, Chunyi, Lu, Songwu, and Wang, Xinbing. Energy-based rate adaptation for 802.11n. In *Proc. of ACM MobiCom* (2012), ACM, pp. 341–352.

[42] Lim, Yeon-Sup, Chen, Yung-Chih, Nahum, Erich M., Towsley, Don, and Gibbens, Richard J. How green is multipath TCP for mobile devices? In *Proc. of ACM AllThingsCellular* (2014), pp. 3–8.

[43] Linux Foundation. Linux advanced routing and traffic control. `http://lartc.org/howto/`.

[44] Lu, Feng, Voelker, Geoffrey M, and Snoeren, Alex C. SloMo: Downclocking WiFi communication. In *Proc. of USENIX NSDI* (2013), pp. 255–258.

[45] Ma, Xiao, Huang, Peng, Jin, Xinxin, Wang, Pei, Park, Soyeon, Shen, Dongcai, Zhou, Yuanyuan, Saul, Lawrence K, and Voelker, Geoffrey M. eDoctor: Automatically diagnosing abnormal battery drain issues on smartphones. In *Proc. of USENIX NSDI* (2013), pp. 57–70.

[46] Mahajan, Ratul, Rodrig, Maya, Wetherall, David, and Zahorjan, John. Analyzing the MAC-level behavior of wireless networks in the wild. In *Proc. of ACM SIGCOMM* (2006), pp. 75–86.

[47] Manweiler, Justin, and Roy Choudhury, Romit. Avoiding the rush hours: WiFi energy management via traffic isolation. In *Proc. of ACM MobiSys* (2011), pp. 253–266.

[48] McCullough, John C, Agarwal, Yuvraj, Chandrashekar, Jaideep, Kuppuswamy, Sathyanarayan, Snoeren, Alex C, and Gupta, Rajesh K. Evaluating the effectiveness of model-based power characterization. In *Proc. of USENIX Annual Technical Conference* (2011).

[49] Nika, Ana, Zhu, Yibo, Ding, Ning, Jindal, Abhilash, Hu, Y. Charlie, Zhou, Xia, Zhao, Ben, and Zheng, Haitao. Energy and performance of smartphone radio bundling in outdoor environments. In *Proc. of WWW* (2015), pp. 809–819.

[50] Nikravesh, Ashkan, Goo, Yihua, Qian, Feng, Mao, Z. Morley, and Sen, Subhrata. An in-depth understanding of multipath TCP on mobile devices: Measurement and system design. In *Proc. of ACM MobiCom* (2016), pp. 189–201.

[51] Paasch, C., and Barre, S. Multipath TCP in the Linux kernel. `http://www.multipath-tcp.org`.

[52] Paasch, Christoph, Detal, Gregory, Duchene, Fabien, Raiciu, Costin, and Bonaventure, Olivier. Exploring mobile/WiFi handover with multipath TCP. In *Proc. of ACM Cellnet* (2012), pp. 31–36.

[53] Paasch, Christoph, Ferlin, Simone, Alay, Ozgu, and Bonaventure, Olivier. Experimental evaluation of multipath TCP schedulers. In *Proc. of ACM SIGCOMM Workshop on Capacity Sharing Workshop* (2014), pp. 27–32.

[54] Paasch, Christoph, Khalili, Ramin, and Bonaventure, Olivier. On the benefits of applying experimental design to improve multipath TCP. In *Proc. of ACM CoNEXT* (2013), pp. 393–398.

[55] Pathak, Abhinav, Hu, Y. Charlie, and Zhang, Ming. Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with Eprof. In *EuroSys* (2012), pp. 29–42.

[56] Pathak, Abhinav, Hu, Y. Charlie, Zhang, Ming, Bahl, Paramvir, and Wang, Yi-Min. Fine-grained power modeling for smartphones using system call tracing. In *EuroSys* (2011), pp. 153–168.

[57] Peng, Qiuyu, Chen, Minghua, Walid, Anwar, and Low, Steven. Energy efficient multipath TCP for mobile devices. In *Proc. of ACM MobiHoc* (2014), pp. 257–266.

[58] Pluntke, Christopher, Eggert, Lars, and Kiukkonen, Niko. Saving mobile device energy with multipath TCP. In *Proc. of ACM MobiArch* (2011), pp. 1–6.

[59] Postel, J. Transmission control protocol.

[60] Qian, Feng, Wang, Zhaoguang, Gerber, Alexandre, Mao, Zhuoqing Morley, Sen, Subhabrata, and Spatscheck, Oliver. Profiling resource usage for mobile applications: A cross-layer approach. In *Proc. of ACM MobiSys* (2011), pp. 321–334.

[61] Raiciu, Costin, Barre, Sebastien, Pluntke, Christopher, Greenhalgh, Adam, Wischik, Damon, and Handley, Mark. Improving datacenter performance and robustness with multipath TCP. In *Proc. of ACM SIGCOMM* (2011), pp. 266–277.

[62] Raiciu, Costin, Niculescu, Dragos, Bagnulo, Marcelo, and Handley, Mark James. Opportunistic mobility with multipath TCP. In *Proc. of ACM MobiArch* (2011), pp. 7–12.

[63] Raiciu, Costin, Paasch, Christoph, Barre, Sebastien, Ford, Alan, Honda, Michio, Duchene, Fabien, Bonaventure, Olivier, and Handley, Mark. How hard can it be? Designing and implementing a deployable multipath TCP. In *Proc. of USENIX NSDI* (2012), pp. 399–412.

[64] Rao, Ashwin, Lim, Yeon-sup, Barakat, Chadi, Legout, Arnaud, Towsley, Don, and Dabbous, Walid. Network characteristics of video streaming traffic. In *Proc. of ACM CoNEXT* (2011), pp. 25:1–25:12.

[65] Rockwell, Peter J., and Davis, Richard A. *Introduction to Time Series and Forecasting.* Springer, 1994.

[66] Roshan, Pejman, and Leary, Jonathan. *802.11 Wireless LAN Fundamentals.* 2010.

[67] Samsung Electronics. Samsung Galaxy S3 Android phone specifications. `http://www.samsung.com/us/mobile/cell-phones/SGH-I747MBBATT`.

[68] Sandvine. Global Internet phenomena report, 1H 2014. `https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/1h-2014-global-internet-phenomena-report.pdf`.

[69] Schulman, Aaron, Navda, Vishnu, Ramjee, Ramachandran, Spring, Neil, Deshpande, Pralhad, Grunewald, Calvin, Jain, Kamal, and Padmanabhan, Venkata N. Bartendr: A practical approach to energy-aware cellular data scheduling. In *Proc. of ACM MobiCom* (2010), pp. 85–96.

[70] Shin, KwangSik, Kim, Jinhyuk, and Choi, Sang Bang. Loss recovery scheme for TCP using MAC MIB over wireless access networks. *IEEE Communications Letters 15*, 10 (Oct. 2011), 1059 –1061.

[71] Simonsson, A., and Furuskar, A. Uplink power control in LTE - Overview and performance. In *Proc. of IEEE VTC* (2008), pp. 1–5.

[72] Stockhammer, Thomas. Dynamic adaptive streaming over HTTP – Standards and design principles. In *Proc. of ACM MMSys* (2011), pp. 133–144.

[73] Tarkoma, Sasu, Siekkinen, Matti, Lagerspetz, Eemil, and Xiao, Yu. *Smartphone Energy Consumption Modeling and Optimization.* Cambridge University Press, 2014.

[74] Vallina-Rodriguez, Narseo, Auçinas, Andrius, Almeida, Mario, Grunenberger, Yan, Papagiannaki, Konstantina, and Crowcroft, Jon. RILAnalyzer: A comprehensive 3G monitor on your phone. In *Proc. of ACM IMC* (2013), pp. 257–264.

[75] Wikipedia. Snapdragon (system on chip). `https://en.wikipedia.org/wiki/Snapdragon_S4#Snapdragon_S4`.

[76] Wischik, Damon, Raiciu, Costin, Greenhalgh, Adam, and Handley, Mark. Design, implementation and evaluation of congestion control for multipath TCP. In *Proc. of USENIX NSDI* (2011), pp. 99–112.

[77] Xu, Fengyuan, Liu, Yunxin, Li, Qun, and Zhang, Yongguang. V-edge: Fast self-constructive power modeling of smartphones based on battery voltage dynamics. In *Proc. of USENIX NSDI* (2013), pp. 43–55.

[78] Yang, F., Amer, P., and Ekiz, N. A scheduler for multipath TCP. In *Proc. of ICCCN* (2013), pp. 1–7.

[79] Yin, Jun, Wang, Xiaodong, and Agrawal, D.P. Optimal packet size in error-prone channel for IEEE 802.11 distributed coordination function. In *Proc. of IEEE WCNC* (2004), pp. 1654–1659 Vol.3.

[80] Zhang, Lide, Tiwana, Birjodh, Qian, Zhiyun, Wang, Zhaoguang, Dick, Robert P., Mao, Zhuoqing Morley, and Yang, Lei. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *CODES+ISSS* (2010), pp. 105–114.