A FRAMEWORK FOR OPTIMIZING DATA TRANSFERS BETWEEN EDGE DEVICES AND THE CLOUD USING COMPRESSION UTILITIES

by

ARMEN A. DZHAGARYAN

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
The Department of Electrical & Computer Engineering
to
The School of Graduate Studies
of
The University of Alabama in Huntsville

HUNTSVILLE, ALABAMA

2016

In presenting this dissertation in partial fulfillment of the requirements for a doctor of philosophy degree from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department or the Dean of the School of Graduate Studies. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this dissertation.

_____ _____

(student signature) (date)

DISSERTATION APPROVAL FORM

Submitted by Armen A. Dzhagaryan in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Engineering and accepted on behalf of the Faculty of the School of Graduate Studies by the dissertation committee.

We, the undersigned members of the Graduate Faculty of The University of Alabama in Huntsville, certify that we have advised and/or supervised the candidate on the work described in this dissertation. We further certify that we have reviewed the dissertation manuscript and approve it in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Engineering.

(Dr. Aleksandar Milenkovic) (date)	Committee Chair
(Dr. Emil Jovanov) (date)	Committee Member
(Dr. Gregg Vaughn) (date)	Committee Member
(Dr. B. Earl Wells) (date)	Committee Member
(Dr. David Coe) (date)	Committee Member
(Dr. Ravi Gorur) (date)	Department Chair
(Dr. Shankar Mahalingam) (date)	College Dean
(Dr. David Berkowitz) (date)	Graduate Dean

ABSTRACT

The School of Graduate Studies
The University of Alabama in Huntsville

Degree Doctor of Philosophy College/Dept. Engineering/Electrical & Computer Engineering

Name of Candidate Armen A. Dzhagaryan
Title A Framework for Optimizing Data Transfers between Edge Devices and the Cloud Using Compression Utilities

An exponential growth of data traffic that originates on edge devices and a shift toward cloud computing necessitate finding new approaches to optimize file transfers. Whereas compression utilities can improve effective throughput and energy efficiency of file transfers between edge devices and the cloud, finding a best-performing utility for a given file transfer is a challenging task. In this dissertation, we introduce a framework for optimizing file transfers between edge devices and the cloud using compression utilities. The proposed framework involves agents running on edge devices and the cloud that are responsible for selecting an effective transfer mode by considering characteristics of transferred files, network conditions, and device performance. The framework agents deployed on a smartphone and a workstation are experimentally evaluated with transfers of varied datasets to and from a local server and cloud instances over networks with varying levels of throughput. The results of the experimental evaluation demonstrate that the framework improves throughput and energy efficiency and reduces costs of file transfers.

Abstract Approval:

Committee Chair _____
(Dr. Aleksandar Milenkovic)

Department Chair _____
(Dr. Ravi Gorur)

Graduate Dean _____
(Dr. David Berkowitz)

To Irina and Svetlana, my dear mother and aunt,
whose love, warmth, and support made me what I am today
and helped me make my dissertation a successful journey.

ACKNOWLEDGMENTS

The work presented in this dissertation would not be possible without the assistance of a number of people who need to be acknowledged. Foremost, I would like to thank my advisor, Dr. Aleksandar Milenkovic, for his continuous counsel and support throughout the entire time. Next, I would like to thank Dr. Mladen Milosevic, who designed *mPowerProfile*, the original tool for energy profiling for mobile platforms. This tool served as a first step in designing *mLViewPowerProfile*, which I used in this research to acquire power traces from mobile devices for preliminary performance and energy efficiency studies of several lossless compression utilities, and for subsequent framework evaluation. I also want to thank Dr. Emil Jovanov, Dr. Wells, and Dr. Seong-Moo Yoo for their guidance and advice during my time as a graduate student. Their courses and projects made me a better engineer and researcher. I would also like to acknowledge all other members and researchers in the LaCASA and mHealth laboratories directed by Dr. Milenkovic and Dr. Jovanov. Finally, I would like to thank Simon Lindley, who has been the initiator of a Ph.D. tuition reimbursement program while being my manager at GRA, Inc.

Most importantly I would like to thank my family, my mother Irina, my aunt Svetlana, and my uncle-in-law Shaun, for their unconditional love and support. I am grateful for their indispensable encouragement and motivation in pursuing my academic goals.

TABLE OF CONTENTS

Page

LIST OF FIGURES xi

LIST OF TABLES xv

CHAPTER

TABLE OF CONTENTS 7

1 INTRODUCTION 1

1.1 Background and Motivation 3

1.2 Data Compression 4

1.3 What Has Been Done? 6

1.4 Main Contributions 7

1.5 Dissertation Outline 8

2 BACKGROUND 10

2.1 Data Transfers between Edge Computing Systems and the Cloud 10

2.2 Lossless Compression Utilities 13

2.2.1 gzip 14

2.2.2 lzop 15

2.2.3 bzip2 15

2.2.4 xz 16

2.2.5 pigz 17

2.2.6 pbzip2 18
2.3 Applications Benefiting from Optimized Data Transfers 18
2.3.1 Software Repositories on Mobile and Desktop Operating Systems 19
2.3.2 File Distribution and Storage Offload 19
2.3.3 HTTP and Web Compression 20
2.3.4 mHealth Applications 21
2.3.5 Scientific and Big Data Offload and Distribution 21
2.4 Experimental Setup 22
2.4.1 Target Platforms and the Cloud 22
2.4.2 Measuring Setup 27
2.4.3 Datasets 39
2.5 Metrics and Experiments 42
2.5.1 Metrics 42
2.5.2 Experiments 45
2.6 The Cases for Intelligent File Transfers 48
2.6.1 Compressed Transfers on the Smartphone 49
2.6.2 Compressed Transfers on Workstations 65
2.6.3 The Case for a Framework for Optimized Data Transfers 82
3 RELATED WORK 85
3.1 Lossless Data Compression on Mobile and Cloud Systems 86
3.2 Power Profiling and Energy Estimation 88

3.2.1 Mobile Devices 88
3.2.2 Workstations 90
4 FRAMEWORK OVERVIEW AND DESIGN 92
4.1 Framework Overview 92
4.1.1 Optimized File Uploads 92
4.1.2 Optimized File Downloads 95
4.2 Framework Components 99
4.2.1 Modeling Uncompressed File Transfers 99
4.2.2 Modeling Compressed File Transfers 119
4.2.3 Collecting Local Compression and Decompression Prediction Data 154
4.2.4 Prediction Data Tables 156
5 FRAMEWORK IMPLEMENTATION 163
5.1 Upload Agents 164
5.2 Download Agents 169
5.3 Re-compression and Server's Long-term Storage 173
5.3.1 Re-compression 173
5.3.2 Optimization 173
6 EVALUATION OF FRAMEWORK IMPLEMENTATION 175
6.1 Evaluation Methods 175
6.1.1 Mobile Device 175
6.1.2 Workstation 176

6.2 Results: Mobile Device 177

6.2.1 Throughput Optimized Transfers 177

6.2.2 Energy Efficiency Optimized Transfers 186

6.3 Results: Workstation 196

6.3.1 Throughput Optimized Transfers 196

6.3.2 Cost Savings 206

7 CONCLUSIONS AND FUTURE WORK 213

LIST OF FIGURES

Figure Page

2.1 Data transfers between mobile devices (a), workstations (b) and the cloud 12

2.2 Block diagrams of an Intel Xeon E3-1240 v2 (a) and an Intel Xeon E5-2650 v2 (b) 25

2.3 Hardware setup for energy profiling of mobile devices under test 29

2.4 Block diagram of the hardware setup for energy profiling of mobile device 29

2.5 Nexus 4 (a) and OnePlus One (b) prepared for energy measurements 30

2.6 *mLViewPowerProfile* user interface 32

2.7 Sample ADB Shell script launched from the workstation to execute on the smartphone 33

2.8 Command script *runGzip.sh* for running a compression task 33

2.9 Current draw during execution of *runGzip6.sh* run-script 35

2.10 likwid-powermeter output for gzip -6 example 36

2.11 Average power traces (PKG, PP0) caputred for gzip -6 example 37

2.12 perf stat output for gzip -6 example 38

2.13 Data flow of the experiments on mobile devices (blue file icons refer to uncompressed files, red file icons refer to compressed files). 46

2.14 Data flow of the experiments on workstation (blue file icons refer to uncompressed files, red file icons refer to compressed files). 48

2.15 Upload throughput speedup for mHealth files on 0.5 MB/s WLAN connection 51

2.16 Upload throughput speedup for mHealth files on 5 MB/s WLAN connection . 53

2.17 Upload energy efficiency improvement for mHealth files on 0.5 MB/s WLAN connection 55
2.18 Upload energy efficiency improvement for mHealth files on 5 MB/s WLAN connection 57
2.19 Download throughput speedup for application, source code, e-book, and map on 0.5 MB/s WLAN connection 59
2.20 Download throughput speedup for application, source code, e-book, and map on 5 MB/s WLAN connection 61
2.21 Download energy efficiency improvement for application, source code, e-book, and map on 0.5 MB/s WLAN connection 63
2.22 Download energy efficiency improvement for application, source code, e-book, and map on 5 MB/s WLAN connection 65
2.23 Upload throughput speedup for upload of wikipages, netcdf, and seqall files to North Virginia on uncapped LAN connection 68
2.24 Upload throughput speedup for upload of wikipages, netcdf, and seqall files to Tokyo on uncapped LAN connection 70
2.25 Upload energy efficiency improvement for upload of wikipages, netcdf, and seqall files to North Virginia on uncapped LAN connection 72
2.26 Upload energy efficiency improvement for upload of wikipages, netcdf, and seqall files to Tokyo on uncapped LAN connection 74
2.27 Download throughput speedup for download of wikipages, netcdf, and seqall files from North Virginia on uncapped LAN connection 76
2.28 Download throughput speedup for download of wikipages, netcdf, and seqall files from Tokyo on uncapped LAN connection 78

2.29 Download energy efficiency improvement for download of wikipages, netcdf, and seqall files from North Virginia on uncapped LAN connection 80
2.30 Download energy efficiency improvement for download of wikipages, netcdf, and seqall files from Tokyo on uncapped LAN connection 82
4.1 System view of optimized data file uploads 94
4.2 System view of optimized data file downloads 97
4.3 System view of optimized data file downloads with on-demand compression ... 98
4.4 Creating 1M file using input device */dev/zero* 104
4.5 Uploads over encrypted WLAN channel (OnePlus One): measured throughputs (a) and energy efficiencies (b) 107
4.6 Downloads over encrypted WLAN channel (OnePlus One): measured throughputs (a) and energy efficiencies (b) 108
4.7 Downloads over unencrypted WLAN channel (OnePlus One): measured throughputs (a) and energy efficiencies (b) 110
4.8 Uploads over encrypted 3G/4G channel (OnePlus One): measured throughputs (a) and energy efficiencies (b) 112
4.9 Downloads over encrypted 3G/4G channel (OnePlus One): measured throughputs (a) and energy efficiencies (b) 113
4.10 Uploads over encrypted LAN channel (Dell PowerEdge T110 II): measured throughputs: North Verginia (a) and Tokyo (b) 115
4.11 Downloads over encrypted LAN channel (Dell PowerEdge T110 II): measured throughputs: North Virginia (a) and Tokyo (b) 116
4.12 Extracting network parameters for uploads: throughput (a) and estimated energy efficiency (b) 119

4.13 Throughput limits: compressed uploads for 5 MB (a), 20 MB (b), and 100 MB (c) files 124

4.14 Throughput limits: compressed downloads for 5 MB (a), 20 MB (b), and 100 MB (c) files 127

4.15 Throughput limits: compressed downloads with on-demand compression for 5 MB (a), 20 MB (b), and 100 MB(c) files 130

4.16 Energy efficiency limits: compressed uploads for 5 MB (a), 20 MB (b), 100 MB (c) files 134

4.17 Energy efficiency limits: compressed downloads for 5 MB (a), 20 MB (b), 100 MB (c) files 137

4.18 Energy efficiency limits: compressed downloads with on-demand compression for 5 MB (a), 20 MB (b), 100 MB (c) files 140

4.19 Compressed upload and download with piping: throughput estimation for 5 MB (a), 20 MB (b), and 100 MB (c) files 144

4.20 Compressed upload and download with piping: energy efficiency estimation for 5 MB (a), 20 MB (b), and 100 MB (c) files 145

4.21 Compressed download with on-demand compression and piping: throughput for 5 MB (a), 20 MB (b), and 100 MB (c) files..... 148

4.22 Compressed download with on-demand compression and piping: energy efficiency for 5 MB (a), 20 MB (b), and 100 MB (c) files..... 149

4.23 Current waveforms for performing local compression using *pigz* -6 on set of varying file sizes – alignment to the starting (a) and ending (b) timestamp 151

4.24 CR and Th.C to change in US – compression (OnePlus One) 155

4.25 Prediction tables for edge device (a) and server/cloud (b)..... 158

4.26 Framwork life cycle (C – client, S – server) 160

4.27 Prediction tables for mobile device (a) and server/cloud (b) 162

5.1 Upload of a text file from the client over 5 MB/s and 0.5 MB/s networks, with throughput (TH) and energy efficiency (EE) modes 165

5.2 SQL query for uploading 1.75 MB file with optimized throughput 166

5.3 SQL query for uploading 1.75 MB file with optimized energy efficiency 168

5.4 Download of text file from the client on 5 MB/s and 0.5 MB/s network throughput, with throughput (TH) and energy efficiency (EE) modes 171

5.5 downloadClient.sh – script for decoding of incoming file based on ID 172

6.1 Upload throughput speedup *Th.FW/Th.UUP* on OnePlus One 179

6.2 Upload throughput speedup *Th.FW/Th.CUP(gzip -6)* on OnePlus One 181

6.3 Download throughput speedup *Th.FW/Th.UDW* on OnePlus One 183

6.4 Download throughput speedup *Th.FW/Th.CDW(gzip -6)* on OnePlus One 185

6.5 Upload energy efficiency improvement *EE.FW/EE.UDW* on OnePlus One 188

6.6 Upload energy efficiency improvement *EE.FW/EE.CUP(gzip -6)* on OnePlus One 190

6.7 Download energy efficiency improvement *EE.FW/EE.UDW* on OnePlus One 193

6.8 Download energy efficiency improvement *EE.FW/EE.CDW(gzip -6)* on OnePlus One 195

6.9 Upload throughput speedup *Th.FW/Th.UUP*: North Virginia (a) and Tokyo (b) 198

6.10 Upload throughput speedup *Th.FW/Th.CUP(gzip -6)*: North Virginia (a) and Tokyo (b) 201

6.11 Download throughput speedup *Th.FW/Th.UDW*: North Virginia (a) and Tokyo (b) 203

6.12 Download throughput speedup *Th.FW/Th.CDW(gzip* -6*)*: North Virginia (a) and Tokyo (b) 205

6.13 Upload cost savings for North Virginia and Tokyo transfers: $.FW vs. $.UUP (a) and $.FW vs. $.CUP(gzip -6) (b) 209

6.14 Download cost saving for North Virginia and Tokyo transfers: $.FW vs. $.UDW (a) and $.FW vs. $.CDW(gzip -6) (b) 211

LIST OF TABLES

Table Page

2.1 Lossless Compression Utilities 14

2.2 AWS EC2 Locations and Server Nodes 26

2.3 Datasets to characterize local (de)compression on mobile devices..... 40

2.4 Datasets to characterize local (de)compression on the workstation 41

2.5 Metrics: Performance 43

2.6 Metrics: Energy 45

5.1 SQL query output sorted for throughput optimized upload @ 5 MB/s 166

5.2 SQL query output sorted for energy efficiency optimized upload @ 0.5 MB/s ... 168

6.1 Overall upload throughput speedup *Th.FW/Th.UUP* on OnePlus One 180

6.2 Overall upload throughput speedup *Th.FW/Th.CUP(gzip* -6*)* on OnePlus One 182

6.3 Overall download throughput speedup *Th.FW/Th.UDW* on OnePlus One..... 184

6.4 Overall download throughput speedup *Th.FW/Th.CDW(gzip -6)* on OnePlus One..... 186

6.5 Overall upload energy efficiency improvement *EE.FW/EE.UUP* on OnePlus One 189

6.6 Overall upload energy efficiency improvement *EE.FW/EE.CUP(gzip* -6*)* on OnePlus One 191

6.7 Overall download energy efficiency improvement *EE.FW/EE.UDW* on OnePlus One..... 194

6.8 Overall download energy efficiency improvement *EE.FW/EE.CDW(gzip* -6*)* on OnePlus One 196

6.9 Overall upload throughput speedup *Th.FW/Th.UUP* (North Virginia) 199

6.10 Overall upload throughput speedup *Th.FW/Th.UUP* (Tokyo) 199

6.11 Overall upload throughput speedup *Th.FW/Th.CUP(gzip* -6*)* (North Virginia) 201

6.12 Overall upload throughput Speedup *Th.FW/Th.CUP(gzip -6)* (Tokyo) 202

6.13 Overall download throughput speedup *Th.FW/Th.UDW* (North Virginia)..... 204

6.14 Overall download throughput speedup *Th.FW/Th.UDW* (Tokyo)..... 204

6.15 Overall download throughput speedup *Th.FW/Th.CDW(gzip* -6*)* (North Virginia) 206

6.16 Overall download throughput speedup *Th.FW/Th.CDW(gzip* -6*)* (Tokyo) 206

CHAPTER 1

INTRODUCTION

Cloud and mobile computing represent emerging trends in modern computing and communication. Cloud computing refers to a computing paradigm where resources such as computing nodes, storage, networks, applications, and services are accessed over the Internet and shared by multiple users and enterprises. Mobile computing refers to a computing paradigm where computing and communication services are delivered via wireless enabled mobile platforms such as smartphones or tablet computers. In this dissertation we refer to all devices that rely on the cloud services reached through the Internet as edge devices. Edge devices broadly speaking encompass mobile devices, workstations, and Internet-of-Things devices.

Mobile and wearable platforms are typically resource-constrained systems with limited processing power, storage capacity, energy, and communication bandwidth. They thus rely on the cloud services for fast data processing and long-term data storage. Data originating on wearable platforms (e.g., health records) and mobile computing devices (e.g., videos, images, documents, and sensor data) are transferred to the cloud. Data originating on wearable platforms often have to be transferred to the paired mobile device before reaching the cloud. Similarly, data stored in the cloud (e.g., documents, applications, movies, maps, messages, commands) are transferred from the cloud to edge devices. It is thus critical to guarantee fast, low-

latency, and high-throughput communication channels between edge devices and the cloud.

Besides data generated on mobile and wearable platforms, a large amount of data is produced and consumed by commercial, academic, and government institutions that process and analyze big data and scientific data using cloud and distributed computing. For example, big data applications may analyze web or user's online activities to make predictions in areas, such as product evaluation and market characterization. Next, increased collaboration and data exchange in scientific communities result in an increased communication between edge devices and the cloud. These new data paradigms pose new challenges associated with cost, security, and storage. Whereas the use of cloud services opens new opportunities in computing and reduces the costs of ownership and operating hardware, the costs associated with the use of cloud services can eventually become prohibitively high for small to medium research and industry groups. Providers of cloud platforms charge utilization fees for using computing resources and data transfer fees for data transfers either to or from the cloud. The specific cloud instance configuration (disk space, tenancy type, network priority, computational power) and location of the cloud instance determine the final fees associated with each instance use and data transfers. Consequently, optimizing data transfers between edge devices and the cloud is also important in the context of scientific and industrial computing.

Lossless data compression can increase communication throughput, reduce latency, save energy, reduce cost, and increase available storage. However, compression tasks introduce additional overhead that may exceed gains by transferring fewer bytes. Compression utilities on mobile and workstation computing platforms differ in compression ratio, compression and decompression speeds, and energy re-

quirements. When transferring data we would ideally like to have an agent to determine whether compressed transfers are beneficial, and if so, to select the most beneficial compression utility. This dissertation introduces a framework for optimizing data transfers between edge (mobile and workstation) devices and the cloud by utilizing compression and decompression utilities.

The rest of the Introduction gives background and motivation (Section 1.1), discusses data compression (Section 1.2), introduces the proposed framework for optimizing data transfers between edge devices and the cloud using compression utilities (Section 1.3), presents the main contributions of the dissertation (Section 1.4), and finally, gives the outline of the dissertation (Section 1.5).

1.1 Background and Motivation

Mobile computing devices such as smartphones, tablets, and e-readers have become the dominant platforms for consuming digital information. According to estimates for 2014 [1], [2], vendors shipped 1.2 billion smartphones, up 28.4% from the prior year, and 216 million tablets. Annual sales of smartphones exceeded those of feature phones for the first time in 2013 [3], totaling 1,807 million mobile devices [4]. The total number of mobile devices shipped in 2014 reached 1,839 million, with ~65% being smartphones.

Global mobile data traffic continues to grow exponentially. A report from Cisco states that the global mobile data traffic grew 69% in 2014 relative to 2013, reaching 2.5 exabytes per month, which is over 30 times greater than the total Internet traffic in 2000 [5]. It is forecast that the global mobile data traffic will grow nearly 10-fold from 2014 to 2019, reaching 24.3 exabytes per month. A report on global mobile economy by GSMA [6] states that the number of global SIM connections and the

number of unique mobile users in 2014 was 7.3 billion and 3.6 billion, respectively. The number of SIM connections and the number of unique users are expected to reach 10 billion and 4.6 billion by 2020, respectively. The share of 3G/4G connections accounted for approximately 40% of active connections by the end of 2014, and it is expected that to reach 70% by 2020.

Scientific computing is the second area where an automated framework for optimized data transfers can be beneficial. Scientific data is often transferred from smaller compute nodes to supercomputer centers with large computational power and speed. However, to harness a large number of distributed computing nodes, scientific data is sometimes moved from centralized data centers to distributed computing nodes. Distributed computing projects such as Folding@home [7] and SAT@home [8] use idle time of distributed computing nodes to perform computations for solving scientific problems in various fields (e.g., science, cryptography, financial sector). There are also several efforts of exploring distributed computing in the present mobile era by employing mobile smartphones as computing nodes [9]. In all cases, data has to be transmitted either from the edge devices to the central node or from a central node to distributed devices via the Internet.

1.2 Data Compression

Data compression is critical in data communication between edge and cloud devices. It can help improve operating time, lower communication latencies, reduce costs, and make more effective use of available bandwidth and storage. The general goal of data compression is to reduce the number of bits needed to represent information. Data can be compressed in a lossless or lossy manner. Lossless compression means that the original data can be reproduced exactly by the decompressor. In con-

trast, lossy compression, which often results in much higher compression ratios, can only approximate the original data. This is typically acceptable if the data are meant for human consumption such as audio and video. However, program code and input, medical data, email and other text generally do not tolerate lossy compression. We focus on lossless compression in this dissertation.

Lossless data compression is currently being used to reduce the required bandwidth during file downloads and to speed up web page loads in browsers. Google's Flywheel proxy [10], Google Chrome [11], Amazon Silk [12], as well as mobile applications Onavo Extend [13] and Snappli [14], use proxy servers to provide HTTP compression for all pages during web browsing. For file downloads, several Google services, such as Gmail and Drive, provide *zip* compression [15] of files and attachments [16]. Similarly, application stores such as Google Play and Apple's App Store use *zip* or *zip*-derived containers for application distribution. Several Linux distributions are also using common compression utilities such as *gzip*, *bzip2*, and *xz* for their software repositories and software packages (*.deb* and *.rpm*). The importance of lossless compression in network data transfers has also been recognized in academia. This research extends and builds on our prior work [17]–[22] and complements earlier studies by Barr and Asanović [23], [24] by developing a framework for optimized data transfers between edge devices and the cloud.

The choice of the compression algorithm, the compression level, and the quality of the implementation affect the performance and energy consumption of compressed data transfers. Performance and energy efficiency of compressed data transfers vary widely for different compression utilities (e.g., *gzip*, *lzop*, *bzip2*, *xz*, *pigz*, and *pbzip2*) and for different compression levels within each utility. In addition, network conditions (network throughput and connection), device characteristics

(processor speed, memory, network interfaces), data file size, and data type, all have a significant impact when determining an optimal mode of data transfers. The default compression utility, *gzip* -6, is rarely the most efficient mode of compressed data transfers, as was shown in our prior work [17]–[20].

1.3 What Has Been Done?

In this dissertation, a framework for optimizing data transfers between edge and cloud computing platforms has been developed. The framework automatically selects a mode of data transfer based on specific factors such as characteristics of data to be transferred (size, type), edge device performance and energy efficiency, and network conditions (throughput and setup time). The framework promise to improve efficiency of data transfers (in throughput, energy efficiency, and cost) in several important classes of applications and uses, such as:

- Mobile applications stores and software repositories
  (e.g. Apple App Store, Google Play [25]);
- File distribution in the cloud (e.g. Dropbox [26], Google Drive [27]);
- HTTP compression (server: NGINX [28], Apache [29]; client: web-browsers);
- Collection and transfer of mHealth data files to and from the cloud;
- Big data and scientific data offload to more computationally intensive distributed centers and cloud instances.

The framework is evaluated in the following environments: (i) a mobile device, using a WLAN network connection and a local server, and (ii) a workstation, using a LAN network connection and the cloud instances provided by Amazon's AWS EC2 cloud. The experimental evaluation demonstrates that the proposed

framework can improve throughput, energy efficiency, and reduce costs of data transfers. Additionally, the framework allows for optimizing transfers for the most critical need of a particular user, human, or machine. A throughput based optimization selects a transfer mode that maximizes effective throughput, whereas energy efficiency based optimization selects a transfer mode that maximizes the energy efficiency of the edge device. This means that edge devices, such as smartphones, will benefit through cost reduction due to lowering data usage on cellular plans when using the throughput mode, and will effectively extend battery life through increased energy efficiency when using the energy-efficient mode. Workstations and servers will benefit from both increased performance and reduced costs of cloud subscription when using the throughput mode and will lower energy consumption, and save energy costs when using the energy-efficient mode of optimization.

1.4 Main Contributions

This dissertation makes the following contributions to the field of energy-efficient computing and communication, to the field of compression and decompression on mobile, workstation and cloud platforms, and finally to the field of measurement-based power profiling:

- Development of analytical models for characterization of effective throughput and energy efficiency during uncompressed and compressed data file transfers.
- Development of analytical models for estimating energy efficiency of uncompressed and compressed data transfers derived from models for effective throughput and device characteristics.

- Development of a framework for optimizing data transfers between edge devices and the cloud by utilizing compression and decompression utilities.
- Implementation of the framework across mobile, workstation, and cloud instances.
- Verification of proposed analytical models for characterization of throughput and energy efficiency during uncompressed and compressed data transfers.
- Evaluation of the framework and quantification of gains in performance and energy efficiency when compared to the uncompressed data transfers and the default compressed data transfers with *gzip* -6 on smartphone and workstation platforms.
- In order to cover feasibility of framework implementation across several applications, the framework evaluation is conducted with varied datasets, representative of selected applications and specific to each platform (ranges of file sizes and types), with several network configurations (WLAN, LAN), and with varied server location (local server and globally spread cloud instances using Amazon's AWS EC2 cloud).
- Development of an experimental environment for automated measurement-based energy profiling of applications running on mobile computing platforms.

1.5 Dissertation Outline

The rest of dissertation is organized as follows. CHAPTER 2 gives background on this research, including compression algorithms and utilities, target com-

puting systems and the cloud, data transfer between computing systems and the cloud, experimental setup, and finally cases for intelligent data transfers to explains fundamental principles of the framework. CHAPTER 3 presents related work by highlighting relevant studies that deal with compression, performance and energy efficiency optimization, and finally power profiling and energy estimating techniques. CHAPTER 4 presents framework overview for upload and download, and introduces framework design and components, including models for estimating throughput and energy efficiency for uncompressed and compressed file transfers, and prediction data tables for characterization of compression utilities. CHAPTER 5 details implementation of the framework, including implementation for upload, download, re-compression of files and server's long-term storage. CHAPTER 6 introduces framework evaluation methods and then presents results of evaluation conducted on the smartphone and workstation when using the framework with performance-based and energy-efficient-base configuration. CHAPTER 7 summarizes the dissertation and discusses possible future work in the area of this research.

CHAPTER 2

BACKGROUND

This chapter covers background on several aspects of this research. Section 2.1 introduces ways for transferring data between edge computing systems and the cloud. Section 2.2 gives details on selected lossless compression utilities and their algorithms to provide understanding on how selected lossless compression utilities work at the basic level. Section 2.3 highlights some of the relevant applications where optimized data transfers can be beneficial for improvements in energy efficiency, throughput, and reducing the costs associated with cloud platforms. Section 2.4 introduces experimental setup used for preliminary studies and framework evaluation, including target platforms and the cloud, measuring setup, and datasets. Section 2.5 describes metrics and experiments. Finally, Section 2.6 makes several cases for optimizing data transfers, including examples of compressed transfers on mobile devices (Section 2.6.1), examples compressed transfers on workstations (Section 2.6.2), and the final case for a framework for optimized data transfers (Section 2.6.3).

2.1 Data Transfers between Edge Computing Systems and the Cloud

Figure 2.1 illustrates file uploads and downloads initiated from (a) a mobile device and from (b) a workstation. A data file can be uploaded to the cloud or downloaded from the cloud uncompressed or compressed. For uncompressed transfers, an

uncompressed file (UF) is uploaded or downloaded over a network connection directly. For compressed uploads, the uncompressed file is first compressed locally on the device, and then a compressed file (CF) is uploaded over the network. For compressed downloads, a compressed version of the requested file is downloaded from the cloud, and then the compressed file is decompressed locally on the edge device (mobile or workstation). In a case when compressed version of the requested file is not available in the cloud, the on-demand compression is performed in the cloud, and then the compressed file is downloaded and decompressed locally on the edge device. Compressed uploads and downloads utilize common compression utilities and compression levels pairs available on the local or remote systems, which can include sequential implementations, such as *gzip, lzop, bzip2* and *xz*, and parallel implementations, such as *pigz* and *pbzip2*.

To evaluate the effectiveness of a networked file transfer, we need to determine the total time to complete the transfer. This time, in general, includes the following components: (i) edge device overhead time; (ii) network connection setup time; (iii) file transmission time; and (iv) cloud overhead time. To measure the effectiveness of data transfers, we use the effective throughput rather than the total transfer time. This metric captures the system's ability to perform a file transfer in the shortest period of time regardless of a transfer mode.

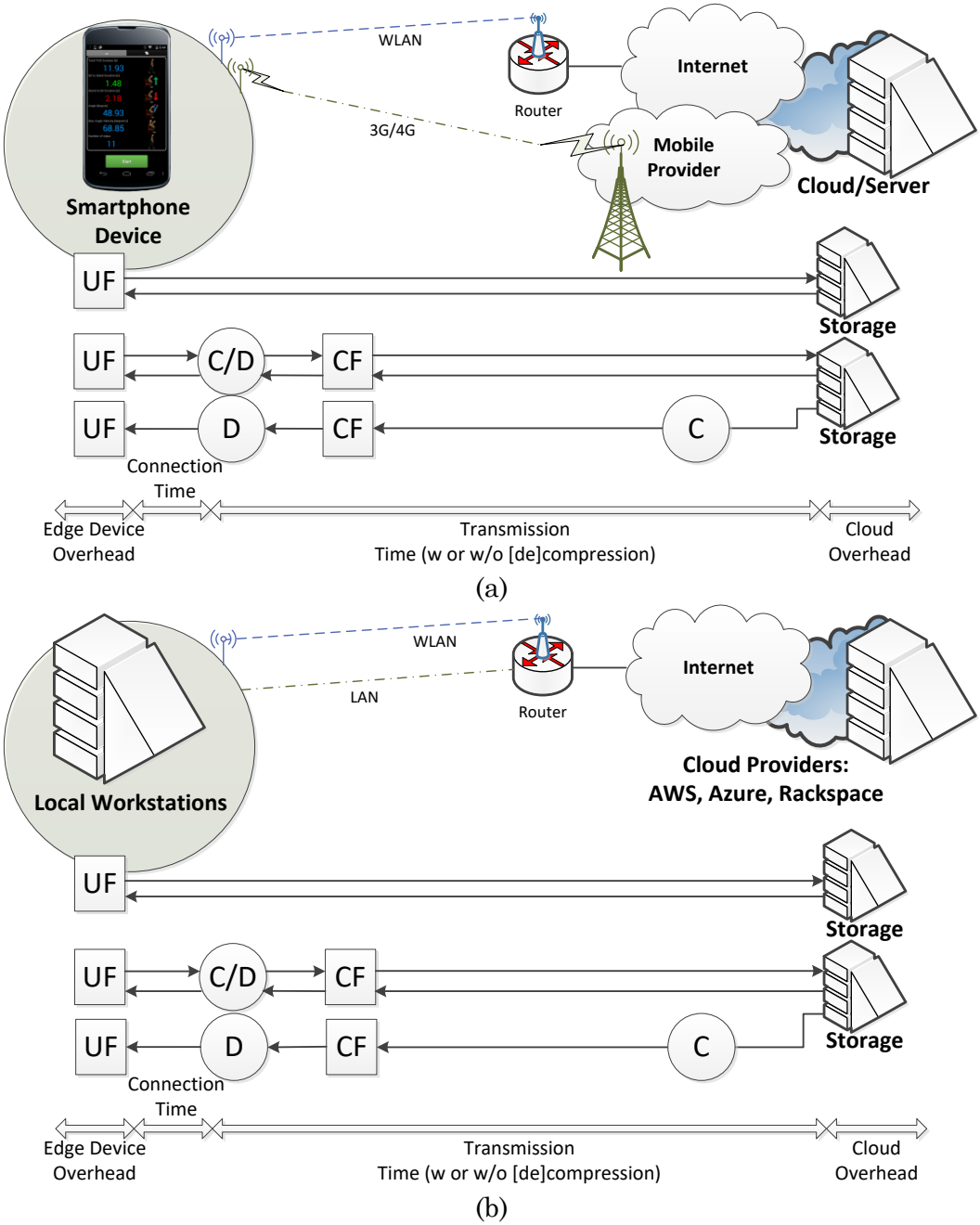


Figure 2.1 Data transfers between mobile devices (a), workstations (b) and the cloud

Another metric of interest for networked file transfers initiated on mobile devices and workstations is energy efficiency. The amount of energy consumed for (de)compression can be a decisive factor in battery-powered mobile devices and cost-

saving factor for workstations transferring large amount of data to and from the cloud. Achieving a higher compression ratio requires more computation and, therefore, more energy, but better compression reduces the number of bytes, thus saving energy when transmitting the data. This metric captures the system's ability to perform a file transfer while consuming the least energy.

2.2 Lossless Compression Utilities

Table 2.1 lists the six lossless compression utilities used in this research along with the supported range of compression levels. The relatively fast *gzip* utility and the slower but better compressing *bzip2* utility are selected due to their widespread use. *lzop* is included because of its high speed. *xz* is gaining popularity and is known for its high compression ratio, relatively slow compression, and fast decompression. As many modern smartphones include multicore CPUs, we also consider *pigz* and *pbzip2*, parallel versions of *gzip* and *bzip2*, respectively. All of these utilities operate at the byte level granularity and support a number of compression levels that allow the user to trade off speed for compression ratio. Lower levels favor speed whereas higher levels result in better compression. Typical compression levels are between -1 and -9, but some utilities provide compression level -0 for no compression, and higher compressions (e.g., *pigz* -11) to provide higher compression ratio at a severe cost in speed. For evaluation of the framework, we consider at least three compression levels for each utility: L – low, M – medium, and H – high. Subsections below describe each utility (*gzip*, *lzop*, *bzip2*, *xz*, *pigz*, and *pbzip2*) and file formats generated by those utilities (*.gz*, *.bz2*, *.lzo*, and *.xz*) in detail.

Table 2.1 Lossless Compression Utilities

Utility	Compression levels (default) (L, M, H)	Version	Magic #	Notes
gzip	1-9 (6) (1, 6, 9)	1.6	0x1f8b	DEFLATE (Ziv-Lempel, Huffman)
lzop	1-9 (3) (1, 6, 9)	1.03	0x894c	LZO (Lempel-Ziv-Oberhumer)
bzip2	1-9 (9) (1, 6, 9)	1.0.6	0x425a	RLE+BWT+MTF+RLE+Huffman
xz	0-9 (6) (1, 6, 9)	5.1.0a	0xfd37	LZMA2
pigz	0, 1-9, 11 (6) (1, 6, 9)	2.3.1	0x1f8b	parallel implementation of gzip
pbzip2	1-9 (9) (1, 6, 9)	1.1.9	0x425a	parallel implementation of bzip2

2.2.1 gzip

gzip [30] implements the DEFLATE algorithm, which is a variant of the LZ77 algorithm [31]. It looks for repeating strings, i.e., sequences of bytes, within a 32 KB sliding window. The length of the strings is limited to 256 bytes. *gzip* uses two Huffman coders, one to compress the distances in the sliding window and another to compress the lengths of the strings as well as the individual bytes that were not part of any matched sequence. The algorithm finds duplicated strings using a chained hash table that is indexed with 3-byte strings. The selected compression level determines the maximum length of the hash chains and whether a lazy evaluation should be used. The evaluated version of *gzip* is 1.6.

gzip compressed files typically end with *.gz* file extension. The *gzip* file format consists of several headers, a body, and a footer. The first header is 10-byte long and contains a magic number (0x1f8b), used to identify a file format, a version number, a timestamp, and an identifier of file system where compression took place. Additional headers contain information such as original file name and are followed by the main body containing a DEFLATE-compressed payload. Finally, the gzip file format ends

with an 8-byte footer, which consists of a CRC-32 checksum and the length of the original uncompressed data.

2.2.2 lzop

lzop [32] uses LZO block-based compression algorithm that favors speed over compression ratio and requires little memory to operate. It splits each block of data into sequences of matches (a sliding dictionary) and non-matching literals, which it then compresses. LZO requires no memory for decompression and requires only 64 KB for compression. The *lzop* implementation on selected mobile device supports only compression levels -1 to -6 while the selected workstation and server platforms support all 9. For the current version of *lzop*, several fast compression levels, including -2, -3, -4, -5, and -6, are equivalent, but open to change in the future releases. Compression levels -7, -8, and -9 offer better compression ratio at the cost of speed. The choice of compression level does not affect the speed of decompression. The evaluated version of *lzop* is 1.03 with the utilized LZO library 2.08.

lzop compressed files typically ends with *.lzo* file extension. The *lzop* file format consists of several headers, a body, and a footer. The first header contains a magic number (0x894c), a version number and a timestamp. The rest of the structure is similar to *gzip* file format, which contains additional headers, main body with compressed blocks, and footer.

2.2.3 bzip2

bzip2 [33] implements a variant of the block-sorting algorithm described by Burrows and Wheeler (BWT) [34]. It applies a reversible transformation to a block of inputs, uses sorting to group bytes with similar contexts together, and then com-

presses them with a Huffman coder. The selected compression level adjusts the block size between 100 KB and 900 KB (with compression levels -1 to -9). *bzip2* requires between 1200 KB and 7600 KB of memory for compression, and 500 KB to 3700 KB for decompression, during compression levels 1 to 9, respectively. The evaluated version of *bzip2* is 1.0.6.

bzip2 compressed file typically ends with *.bz2* file extension. The *bzip2* file format consists of a header, a main body, and a footer. The header is 4-byte long and contains a magic number (0x425A – 'BZ'), a version number ('h' for *bzip2* with Huffman coding), and selected compression level. The header is followed by zero blocks plus compressed blocks containing the Huffman-compressed payload. Similar to *gzip*, *bzip2*'s footer contains a 32-bit CRC checksum.

bzip2 has a number of implementations, including *micro-bzip2*, *pbzip2* (Subsection 2.2.6), *bzip2smp*, *smpbzip2*, *pyflate*, *lbzip2*, *mpibzip2*, *jbzip2*, *DotNetZip*, and several others. The independent nature of *bzip2* compressed blocks allows for parallel decompression since blocks do not depend on one another. Thus, parallel implementations, such as *pbzip2*, can decompression *bzip2* generated compressed files in parallel.

2.2.4 xz

xz [35] is based on the Lempel-Ziv-Markov chain compression algorithm (LZMA) developed for the 7-Zip utility [36]. It uses a large dictionary to achieve high compression ratios and employs a variant of LZ77 with special support for repeated match distances. The output is encoded with a range encoder, which uses a probabilistic model for each bit (rather than whole bytes) to avoid mixing unrelated bits, i.e., to boost the compression ratio. The memory usage of *xz* is substantially higher than

with other compression utilities, ranging from a few MB to several GB. Average memory usage for compression ranges from 30 MB to 674 MB with compression levels 0 to 9. Average memory usage for decompression needs only 5% to 20% of the memory used by the compressor, and ranges from 1 MB to 65 MB. The evaluated version of *xz* is 5.1.0alpha that utilizes the same version of *liblzma* library.

xz compressed files typically end with *.xz* file extension. The *xz* file format consists of a header, a main body, and a footer. The magic number used to identify *xz* generated compressed files is 0xfd37 and it is stored in the first header as with *gzip* and *lzop* file formats.

2.2.5 pigz

pigz [37] is a parallel version of *gzip* for shared memory machines that is based on pthreads. It breaks the input up into 128 KB chunks and concurrently compresses multiple chunks. The compressed data are outputted in their original order. Decompression operates mostly sequentially (using single main thread), however, three additional threads are created for reading, writing, and checksum calculation [38], which can speed up overall decompression. By default, *pigz* will detect the available number of CPU cores and threads on the system and adjust itself to that value. However, the number of compress threads and block size can also be changed by utilizing options *–p* and *–b*. For example, *–p 1* and *–dp 1* avoids the use of additional threads entirely. The evaluated version of *pigz* is 2.3.1.

pigz compressed files typically end with the same *.gz* file extension as *gzip* compressed files. The *pigz* file format is similar to that of *gzip* and consists of similar headers and footer. The main header contains the same magic number (0x1f 8b) used by *gzip* file format, a version number, and a timestamp. Additional headers and

the body contain the original file name and the DEFLATE-compressed payload, respectively. Finally, *pigz* ends with 8-byte footer, which consists of a CRC-32 checksum and the length of the original uncompressed data. Having the same headers and footers allows *gzip* to decompress *pigz* generated compressed files sequentially, and vice versa, allows *pigz* to decompress *gzip* generated compressed files in parallel (with additional threads for reading, writing and checksum calculation).

2.2.6 pbzip2

pbzip2 [39] is a multithreaded version of *bzip2* that is based on pthreads. It works by compressing multiple blocks of data simultaneously. Similar to *bzip2*, the selected compression level adjusts the block size between 100 KB and 900 KB (with compression levels -1 to -9). The resulting blocks are then concatenated to form the final compressed file, which is compatible with *bzip2*. Decompression is also parallelized. The evaluated version of pbzip2 is 1.1.9.

pbzip2 compressed files typically end with the same *.bz2* file extension as *bzip2* compressed files. The *pbzip2* file format is similar to that of *bzip2* and consists of a similar header, main body, and a footer. The header contains the same magic number (0x1f8b) used by *bzip2*, a version number and a timestamp.

2.3 Applications Benefiting from Optimized Data Transfers

There are a number of applications on mobile and workstation platforms where optimized data transfers can yield improvements in throughput, energy efficiency, as well as in the reduction of costs, associated with utilization and data transfer fees on cloud platforms. These types of applications include (i) application stores and software repositories (e.g., Apple Store, Google Play), (ii) file distribution

and storage offload (e.g., DropBox, Google Drive), (iii) HTTP compression, (iv) mHealth data transfers between the edge devices and the cloud, and (v) computational and storage offload of scientific and big data to the cloud.

2.3.1 Software Repositories on Mobile and Desktop Operating Systems

Application stores and software repositories is an important area for mobile devices and workstations alike. Some of the main software repositories for mobile devices include official Google Play [25] and iOS App Store, as well as unofficial repositories such as F-Droid [40], which is a free and open-source alternative to Google Play. For workstations running Linux, Debian and RPM packages make up software repositories on a number of Debian and Fedora derived Linux distributions. Packages in those software repositories are managed by package manager systems for installation, removal, and updating programs on local workstations.

Currently, mobile software packages (*.apk*, *.ipa*) utilize *gzip* compression, and Linux *.deb* and *.rpm* packages can support *gzip, xz*, or *bzip2* compression (including independent compression of internal subdirectories; e.g., *control.tar.gz* and *data.tar.xz* for Debian package). In all cases, the use of compression is static and its download performance cannot be improved under different of network parameters, different types of data stored in the containers, or different types of devices performing the download.

2.3.2 File Distribution and Storage Offload

The second important area for mobile and workstation platforms within the cloud paradigm is file distribution and storage offload, where files are distributed across multiple devices or offloaded to compensate for limited internal or local stor-

age. File distribution is driven by the need of having files stored and synchronized across multiple devices, starting from individual levels, such as across multiple personal devices in a household, to various levels in enterprises, workgroups, and research groups. Storage offload is driven by the limitations of internal flash storage on mobile devices, and by the limitations of local storage with workgroups and research groups generating a large amount of data. The use of cloud services is driven by cost savings achieved by leasing computing and/or storage resources, instead of direct purchase and maintenance of hardware resources.

Examples of file distribution services include commercial systems such as Dropbox [26], and open source projects such as ownCloud [41], and its derivative NextCloud [42]. Similar services are offered by a number of cloud providers. Selected file distribution servers, such as Google Drive and Gmail client, provide *zip* compression [15] for files and attachments [16] on downloads.

2.3.3 HTTP and Web Compression

Lossless data compression is also being used to reduce the required bandwidth during file downloads and to speed up web page loads in the browsers. Currently, HTTP compression is limited to *gzip* implementations provided by the modules built into Apache and NGINX HTTP web-server projects [28], [29], and by support on selected web browser clients (HTTP web clients). To compensate for possible lack of HTTP compression support on some servers, Google's Flywheel proxy [10], Google Chrome [11], Amazon Silk [12], as well as the mobile applications Onavo Extend [13] and Snappli [14], use proxy servers to provide HTTP compression for all pages during web browsing.

2.3.4 mHealth Applications

With a continual growth of the number and type of wearable devices and their market proliferation, the amount of data that needs to be transferred between personal devices and the cloud is growing exponentially. mHealth data generated on wearable devices is first retrieved by personal devices (e.g., via Bluetooth paired connection), and then transferred to the cloud through 3G/4G or WLAN connection. Examples of existing solutions include a number of activity trackers such as Fitbit trackers, Garmin trackers, Microsoft Band, and Intel Basis Peak. They all perform regular offload and synchronization of collected raw mHealth data to the cloud through the use of paired device (e.g., smartphone). Such offload can take place either continuously, when the personal device is within their proximity, or periodically once the wearable device is reconnected with the personal device, for example after activity takes place. Typically, once raw data is uploaded to the cloud, processed results are later downloaded to the mobile devices through appropriate applications or by any devices through appropriate web-portals. Retrieved data from wearable devices typically come in a proprietary binary format (such as *.dat* file format for Zephyr BioHarness 3) or in text format (such as .CSV or XML file formats). The mobile device and the corresponding application for the particular wearable device can send wearable mHealth data with or without compression to the cloud.

2.3.5 Scientific and Big Data Offload and Distribution

Besides storage and data distribution, another important area for workstation platforms within industry, workgroups and research groups includes offload and distribution of scientific and big data for purposes of computational speedup. Offload and distribution of such data can be done either to the cloud (using cloud

providers) or to computationally intensive collaborating centers. Cloud providers such as Amazon's AWS, Microsoft Azure, Rackspace address this need by leasing cloud instances designed for specific computational needs.

2.4 Experimental Setup

For this dissertation, several computer systems with varying hardware complexity are selected to support framework implementations and evaluations, representing typical mobile and workstation platforms that may benefit from optimized data transfers. Subsection 2.4.1 describes the main characteristics of the target platforms, including the mobile devices, workstations, as well as the cloud platform used in the cloud experiments. Subsection 2.4.2 describes our measurement setup and the way the energy consumed on mobile and workstation systems is calculated. Finally, subsection 2.4.3 describes datasets selected to be representative of each target platform for the preliminary studies and framework evaluation.

2.4.1 Target Platforms and the Cloud

2.4.1.1 Mobile Devices

For mobile devices, Google's Nexus 4 [43] and OnePlus One [44] are used in experimental evaluation. The Google's Nexus 4 smartphone is powered by a Qualcomm's Snapdragon S4 Pro (APQ8064) SoC [45] that includes a quad-core ARM processor running at up to 1.512 GHz clock and an Adreno 320 graphics processor [46]. Nexus 4 has 2 GB of RAM memory and 16 GB of built-in internal storage. It uses a 4.7 inch display, and includes a 1.3 megapixel front-facing camera and an 8 megapixel rear-facing camera. It supports a range of connectivity options including

WLAN 802.11n, Bluetooth 4.0, USB, HDMI, and several cellular network protocols such as GSM/EDGE/GPRS, 3G UMTS/HSPA+/DC-HSP+, and HSDPA+. Nexus 4 runs Android version 4.3.2 (Jelly Bean). In some cases, an upgrade to Android may be beneficial to (i) support applications and setups not readily available on native Android, and (ii) to further automate performance and energy measurements. Our alternative smartphone setup requires flashing the smartphone with CyanogenMod version 10.2 [47], an open-source operating system for smartphones and tablet computers based on official releases of Android that includes third-party software.

The OnePlus One smartphone is powered by a Qualcomm Snapdragon 801 (MSM89734AC) SoC that features a quad-core ARM-based Krait 400 processor running at up to 2.5 GHz clock frequency, an Adreno 330 graphics processor, and 3 GB of RAM memory. OnePlus One supports a range of communication protocols including WLAN 802.11 a/b/g/n/ac, Bluetooth 4.1, USB, HDMI, and several cellular network protocols such as GSM/HSPA/LTE. OnePlus One has preinstalled CyanogenMod version 12.1 that includes third-party software modules needed for power and clock frequency management.

2.4.1.2 Workstations and Servers

For the workstation and server platforms, a Dell PowerEdge T110 II with CentOS 6.5 and Dell PowerEdge T620 with Ubuntu 15.10 are used. The T110 II machine is used for evaluation of the framework implementation for a workstation over LAN interface while using several scientific datasets and a number of remote locations in the cloud platform. The T620 machine is used as a local server platform for evaluation of the framework implementation on mobile devices (OnePlus One).

Dell PowerEdge T110 II features a quad-core Intel Xeon E3-1240 v2 processor which consists of a single 32 nm CMOS monolithic die with four 2-way threaded physical processor cores for a total of 8 logical processor cores, a shared 8 MB L3/LLC cache memory, an integrated memory controller, PCI and DMI interfaces, and a graphics processor. The processor is based on Sandy Bridge architecture and supports 15 frequency steps ranging from 1.60 GHz to 3.40 GHz on each core. The system memory is 8 GB DIMM DDR3 synchronous at 1333 MHz (0.8 ns). The secondary storage includes an ATA hard disk with a capacity of 2 TB. The workstation includes a gigabit network interface, a USB controller, audio and video interfaces.

Dell PowerEdge T620 features two Intel Xeon E5-2650 v2 processors based on Ivy Bridge-EP architecture. Each Xeon E5-2650 v2 processor consists of a single 22nm CMOS monolithic die with eight 2-way threaded physical processor cores for a total of 16 logical processor cores, a shared 20 MB L3/LLC cache memory, an integrated memory controller, and QPI, PCI and DMI interfaces. Each processor supports 15 frequency steps ranging from 1.2 GHz to 2.6 GHz for each core, and features Intel Turbo Boost Technology 2.0 [48] which allows boosting of all active cores up to 3.4 GHz, if specification limits of power, current, and temperature are met. The workstation includes 64 GB of memory, 3 TB of storage, a gigabit network interface, a USB and audio/video interfaces, and NVIDIA PCI Express graphics card.

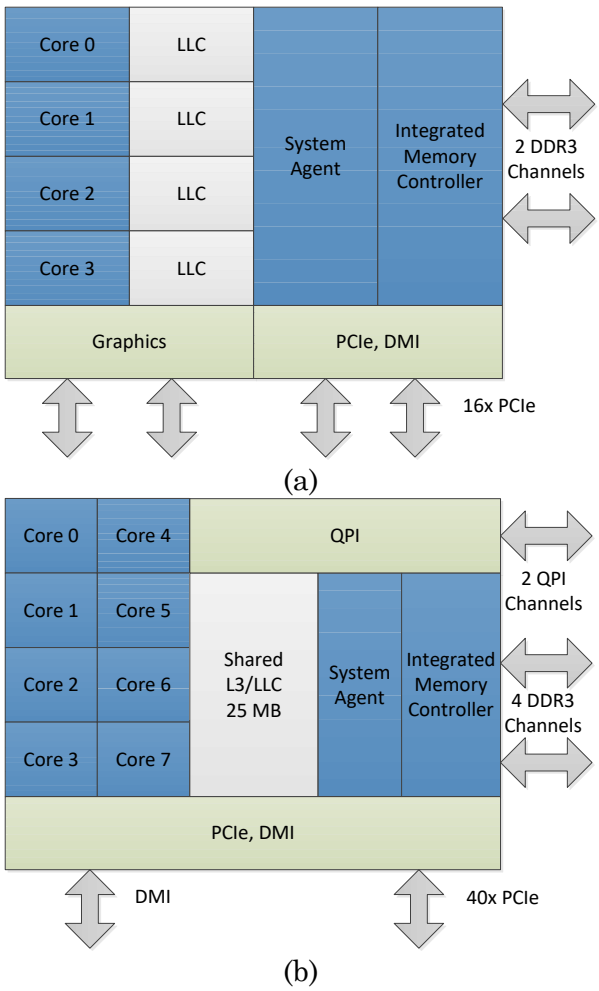


Figure 2.2 Block diagrams of an Intel Xeon E3-1240 v2 (a) and an Intel Xeon E5-2650 v2 (b)

Processors on both workstations also feature a system agent (Figure 2.2) which encompasses a module responsible for power management called Package Control Unit (PCU). The PCU connects to individual processor cores and other functional blocks via power management agents that collect information about power consumption and junction temperature. The PCU runs firmware that constantly monitors power and thermal conditions and performs various power-management functions, e.g., turns on or off a processor cores or portions of the LLC cache or dy-

namically scales voltage and frequency. This allows the use of *likwid* lightweight performance tools [49], [50] to perform power and energy measurements. The *likwid-powermeter* tool interfaces the power meter and outputs power measurements in joules and watts. Intel researchers demonstrated that this on-chip resource gives estimates for the energy and power that are within several percentages of those acquired by the actual power measurements [51].

2.4.1.3 Cloud Computing Platforms

To facilitate distributed computing nodes for this research, we use the Amazon's AWS Elastic Cloud Computing (EC2) platform. AWS EC2 provides computational and storage resources across a number of global locations. The cloud instances in North Virginia and Tokyo are created using *m4.xlarge* Linux instance type – a compute and memory optimized instance with 2.3 GHz Intel Xeon E5 (Broadwell) or 2.4 GHz Intel Xeon E5 (Haswell) processor, 64 GB of memory, and enhanced network-priority in the cloud. Table 2.2 describes the cloud instances, by specifying their instance type, the location, and the distance in miles.

Table 2.2 AWS EC2 Locations and Server Nodes

Instance type	Location	Distance (miles)
m4.xlarge	US East (North Virginia)	600
m4.xlarge	Asia Pacific (Tokyo)	7,000

The pricing model of AWS EC2 breaks into free-tier, on-demand, and reserved instances. The AWS's free-tier is designed for new customers who receive one

of the least expensive compute instances, t2.micro, additional 30 GB of block storage, 15 GB of bandwidth, and 1 GB of regional data transfers each month, free of charge for the first year. The one-demand pricing model charges instance-type specific and region-specific prices on a per-hour basis. Additionally, instance pricing also depends heavily on their number of CPUs, available memory, storage, and network priority and bandwidth. For example, for the North Virginia region, general purpose t2.nano, t2.large, m4.xlarge, and m4.4xlarge instances cost $0.0065, $0.104, $0.239, and $0.958 per hour, respectively. Other specific instances, such as compute-optimized and GPU specific instances cost more than the general purpose instances. Different regions have a different and sometimes higher pricing. For example, for Tokyo region, the same general purpose instances cost $0.01, $0.16, $0.348, and $1.391 per hour, respectively. Reserved instances pricing provides some savings to on-demand hourly costs when a customer chooses to do partial or upfront payments together with a 1-year or 3-year contract. Besides hourly pricing costs for the selected hardware, AWS EC2 also charges region-specific, per GB, pricing for data transfers going out from Amazon EC2. Transfers between AWS instances are free of charge.

2.4.2 Measuring Setup

2.4.2.1 Power Profiling of Mobile Devices

Our setup for measuring the energy consumed on the smartphone, shown in Figure 2.3, consists of an NI PXIe-4154 battery simulator [52], the smartphone, and a workstation. Figure 2.4 shows a block diagram of the setup including main components and communication channels between them. The battery simulator, a special-

ized programmable power supply, resides inside an NI PXIe-1073 chassis [53], which is connected to an MXI-Express Interface card inside the workstation. The battery simulator is used (i) to power the smartphone through probes on channel 0 by providing 4.1 volts, thus bypassing the actual smartphone battery, and (ii) to measure the current drawn by the smartphone while running applications. The battery simulator is optimized for powering devices under test, including cellular handsets, smartphones, tablets, and other mobile devices. Its +6 V, ±3 A Channel 0 is designed to simulate a lithium-ion battery cell's transient speed, output resistance, and 2-quadrant operation (source/sink). Acting as a data acquisition system (DAQ), the battery simulator samples the current drawn on its channels in terms of samples per second (S/s) with a configurable sampling frequency of up to 200,000 S/s and a sensitivity of current measurements of 1 μA. This means that for the processor on Nexus 4 running at its maximum clock frequency of 1.512 GHz, we can sample the current every 7,560 CPU clock cycles.

To prepare the smartphones for energy profiling, their underlying plastic shields are removed or modified to reveal connections on their motherboards and daughter boards as shown in Figure 2.5. The smartphones' batteries are removed, and power connectors from the battery simulator are connected instead (for OnePlus One, internal USB cable is rewired to battery connector). Connectors to smartphone components such as LCD display, touchscreen, USB, and others can be easily unplugged during power profiling, thus enabling selective profiling that excludes energy consumed by these components.

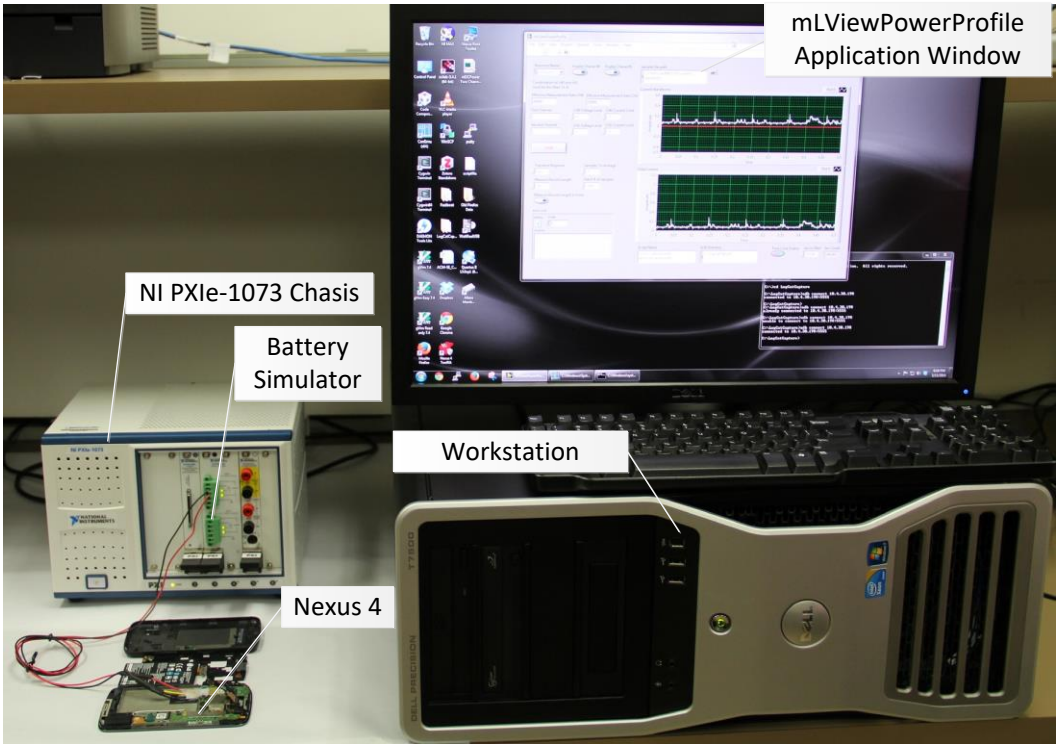


Figure 2.3 Hardware setup for energy profiling of mobile devices under test

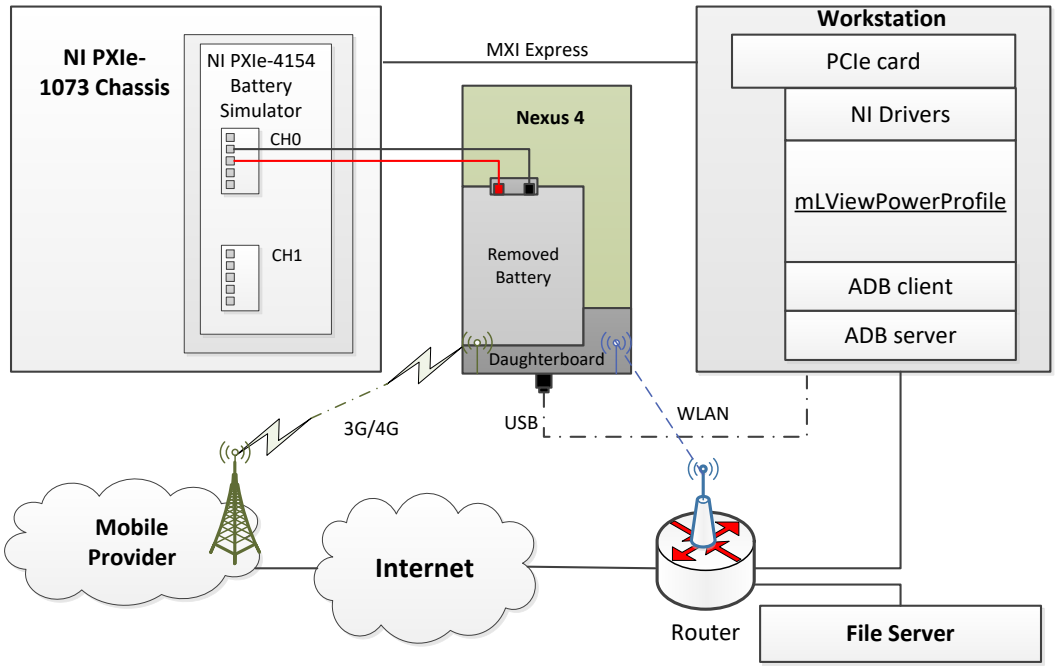


Figure 2.4 Block diagram of the hardware setup for energy profiling of mobile device

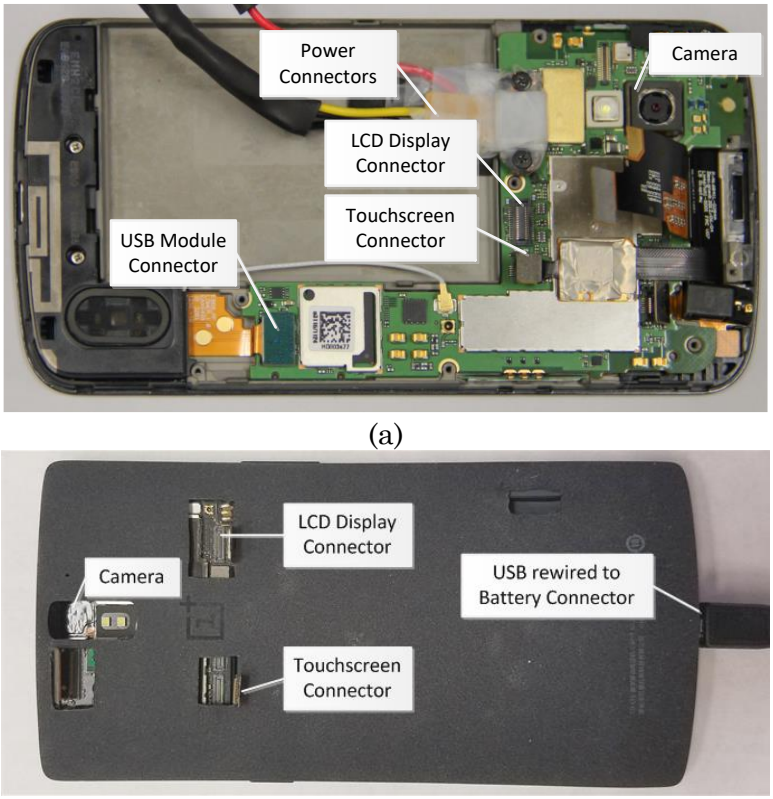


Figure 2.5 Nexus 4 (a) and OnePlus One (b) prepared for energy measurements

The workstation is a Dell T7500 Precision with an Intel Xeon processor, 12 GB of system memory, running the Windows 7 Pro operating system. It runs *mLViewPowerProfile*, our custom software tool for automated capturing of power traces and evaluating energy efficiency of applications running on mobile computing platforms. *mLViewPowerProfile* interfaces (i) the smartphone to manage activities and applications on the smartphone that are being profiled and (ii) the battery simulator to configure the channel and collect the current samples. The communication with the smartphone is carried out over the Android Debug Bridge (*adb*) [54]. *adb* is a client-server program that includes the following components: a client, which runs on the workstation; a server, which runs as a background process on the work-

station; and a daemon, which runs on the smartphone. *adb* can connect to the smartphone over a USB link or over a WLAN interface.

Figure 2.6 shows the *mLViewPowerProfile*'s graphical user interface. A user configures the channels of the battery simulator. This involves setting the voltage and the current limits, the sampling frequency, the transient time, as well as software driver parameters that control fetching the current samples from the battery simulator. *mLViewPowerProfile* can average the maximum sampling rate from the battery simulator by the value set in the graphical user interface. We experimented with different sampling frequencies in the range of 10,000 S/s to 200,000 S/s and evaluated their impact on the energy calculations. We found the energy calculated using 20,000 S/s is within 1% of the energy calculated using the maximum sampling rate of 200,000 S/s, so for our experiments we choose to average 10 samples, thus recording 20,000 S/s in a user-specified file (*appsSamples.txt*).

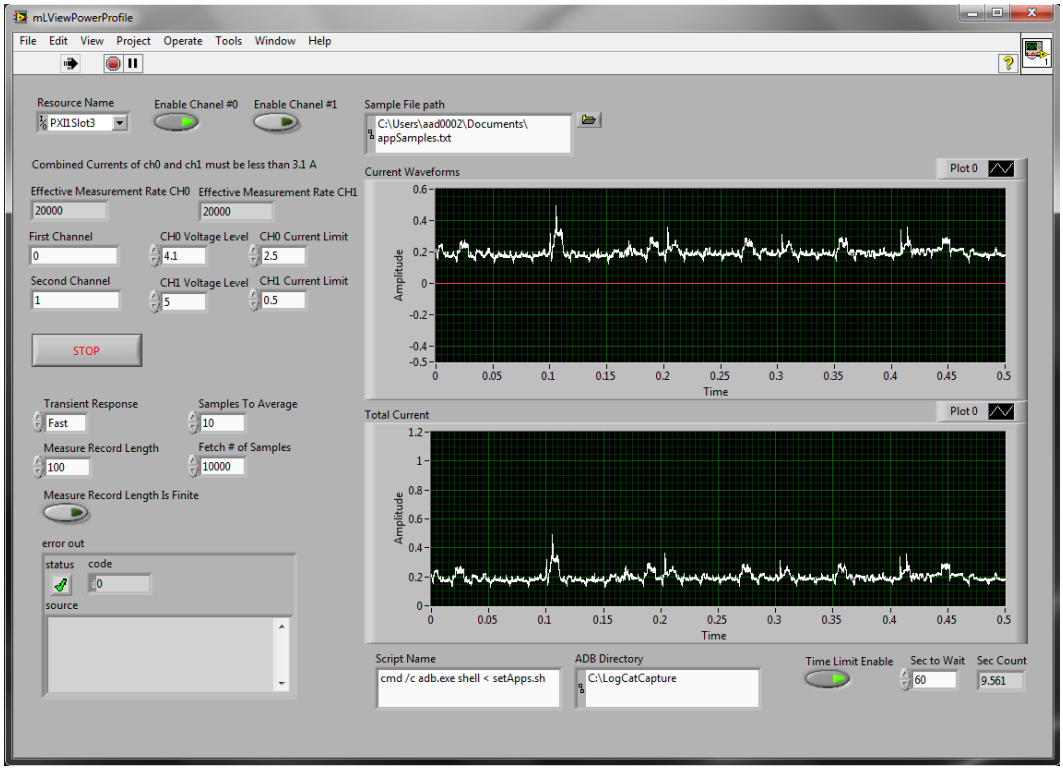


Figure 2.6 *mLViewPowerProfile* user interface

To run a compression or a decompression task on the smartphone, a sequence of *adb* commands is launched from the workstation to be executed on the smartphone as shown in Figure 2.7. The third line executes one of command scripts which are prepared in advance and placed in a working directory of the smartphone. A sample command script with commands for invoking *gzip* compression with default compression level -6 is shown in Figure 2.8. The execution of a (de)compression task is typically preceded and trailed by a 5-second delay (head and tail delays) during which the smartphone is in an idle state. The (de)compression task is wrapped by commands that take time stamps corresponding to the moments when the task is launched and completed. These times are used to determine the task execution time

as well as to identify the appropriate current samples logged on the workstation to calculate the energy consumed by the task.

```
1. su                                   # start as superuser
2. cd /data/working                     # move to working directory
3. nohup ./runGzip.sh &                 # start compression tasks
4. exit                                 # exit superuser session
5. exit                                 # exit adb
```

Figure 2.7 Sample ADB Shell script launched from the workstation to execute on the smartphone

```
1. path="data/working/datainput"              # input location
2. file="totalInput"                          # filename
3. filePath="$path/$file.tar                  # complete file path
4. # run gzip compression task
5. sleep 5;                                   # sleep for 5 seconds
6. cat $EPOCHTIME >> $path /timestamps.txt;   # starting timestamp
7. gzip -c6 $filePath >/dev/null;             # compress input file
8. cat $EPOCHTIME >> $path /timestamps.txt;   # ending timestamp
9. sleep 5                                    # sleep for 5 seconds
```

Figure 2.8 Command script *runGzip.sh* for running a compression task

Figure 2.9 shows the measured current drawn by the Nexus 4 during the execution of the example command script from Figure 2.8. The head and tail delays are 5 seconds each and the compression task takes roughly 17 seconds. The top graph in the figure shows the current drawn during the experiment as it is used in our energy calculations. The bottom graph shows the filtered signal, provided here only to enable easier visual inspection of the changes in the current drawn during program execution. The Nexus 4 with all unnecessary services turned off (LCD disconnected, GPS and WLAN interfaces turned off) draws ~11 mA (I_{IDLE} = 11 mA). The start of

the compression task is marked by a step increase in the current drawn of ~270 mA to 280 mA, the current remains high during the compression, and goes back to the idle level after the compression has terminated. The number of samples during the execution of a compression utility is $n = T.C \times SF$, where $T.C$ is the compression time for a given file and SF is the sampling frequency of the battery simulator (with respect to the number of recorded samples). The total energy consumed ($ET.C$) is calculated as shown in Equation (2.1), where V_{BS} is the supply voltage on the battery simulator (V_{BS} = 4.1 V) and each I_j is a current sample during compression.

In addition to $ET.C$, we also calculate the overhead energy of the compression task alone, $ET.C(0)$, which excludes the energy needed to run the platform when idle. This overhead is calculated as in Equation (2.2). We similarly calculate the total energy and the overhead energy for decompression tasks using the decompression time $T.D$ instead of the compression time $T.D$.

$$ET.C = V_{BS} \cdot \frac{1}{SF} \sum_{j=1}^{n} I_j \tag{2.1}$$

$$ET.C(0) = ET.C - I_{idle} \cdot V_{BS} \cdot T.C \tag{2.2}$$

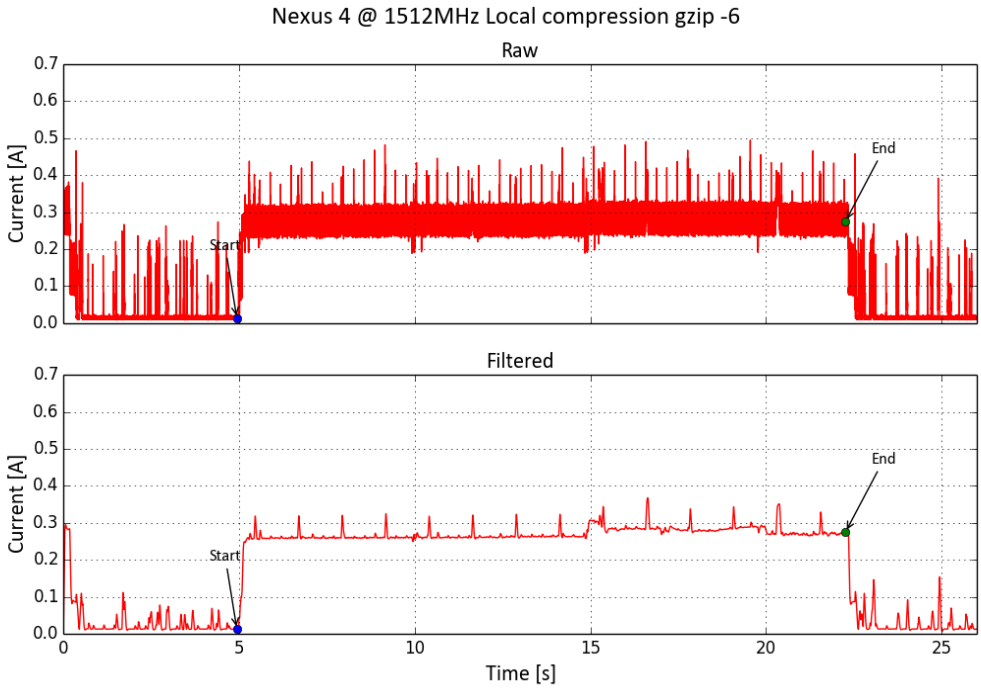


Figure 2.9 Current draw during execution of *runGzip6.sh* run-script

In addition to Linux *time* command, the *Android logging system* provides a mechanism for collecting and viewing system debug output [55]. Custom log messages can be generated with functions declared using *<android/log.h>* include file inside C++ applications compiled with Android NDK tools. The details of power profiling with the utilization of Android logging system is covered in our previous studies on energy profiling of mobile platforms [56], [57].

2.4.2.2 Power Profiling of Workstations

Since both workstations are based either on Intel's Sandy and Ivy bridge processor architecture with RAPL interface, *likwid* tools are used to perform energy profiling of running applications. Additionally, by using Performance Application Programming Interface [58], [59] (PAPI) and *rapl_plot*, *RAPL* energy usage can be

traced and plotted. To get actual performance events, *perf tool* from *linux-tools* package is used.

The *likwid* lightweight performance tools are a collection of simple command line tools for processor topology, affinity and performance profiling and benchmarking. It supports Intel and AMD processors. This tool suite (version 4.1.2) consists of several tools for reading system topology, hardware performance counters, and use of RAPL for energy estimation on Intel processors. The tool which is used for power profiling is *likwid-powermeter*. A typical example of running *likwid-powermeter* on the workstation is shown in Figure 2.10. The first line creates a script file cmd.sh that performs a local compression task using *gzip* -6. To get energy estimates, the *likwid-powertool* is run with the script file as a parameter. The *likwid-powertool* reports the conditions and the energy estimates (from line 3 to line 12). It shows the current clock frequency, the processor core id (CoreId 0) on which the task is run, the execution time, and the energy consumed in Joules for the entire task (181.929 Joules) and the average power consumption (16.3272 Watts).

```
1. echo "gzip -fc -6 /opt/aws/Mdo.seq.all > /dev/null" > cmd.sh
2. ~$ likwid-powermeter ./cmd.sh
3. -------------------------------------------------------------
4. CPU name:       Intel Core SandyBridge processor
5. CPU clock:      3.39 GHz
6. -------------------------------------------------------------
7. Measure on CoreId 0
8. ../md.sh
9. Runtime: 7.81183 s
10.   Domain: PKG
11.   Energy consumed: 158.194 Joules
12.   Power consumed: 20.2505 Watts
```

Figure 2.10 likwid-powermeter output for gzip -6 example

To obtain power tracing with *RAPL* interface, *rapl_plot*, which uses PAPI to poll *RAPL*, is started in background to initiate 10Hz logging of PKG (processor package) and PP0 (cores) energy values and *gzip* -6 example is repeated. Figure 2.11 shows average power traces (PKG and PP0) capture with *rapl_plot* tool when running *gzip* -6.

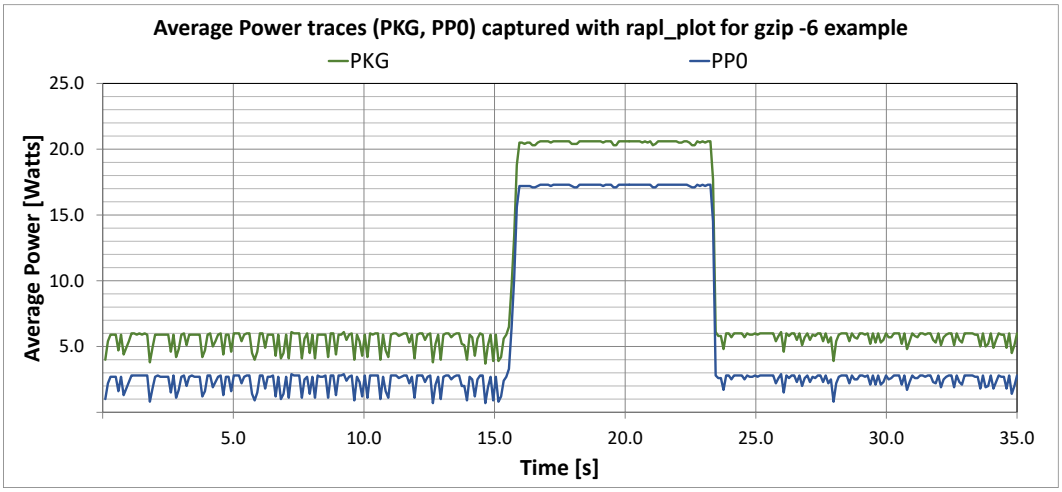


Figure 2.11 Average power traces (PKG, PP0) caputred for gzip -6 example

To conduct a systematic and an autonomous way of running tasks, a bash script is created which rewrite cmd.sh file, executes *likwid-powermeter* and parses output of *likwid-powermeter* for energy and time for each (de)compression task using AWK scripting repeatedly. The output of bash script produces two formatted text files, one with energy values and another with time of execution values for (de)compression tasks.

To access supported performance events, perf tool [60] from *linux_tools* package is used. A typical example of running *perf stat* on the workstation is shown in Figure 2.12. The first line creates the same script file cmd.sh to perform a local com-

pression task using *gzip -6*. To read default performance counters, the *perf stat* is run with the script file as a parameter. The *perf stat* reports a number of performance events, including task-clock, number of context-switches, number of page-faults, number of cycles and instructions, and number of branches and branch-misses. For example, performance counters of executed task reported 8 cpu-migrations, 211 page-faults, 29,348,483,968 cycles, and 8,678,562,885 branches. A complete list of pre-defined performance events supported on the current work-station, can listed by running *perf list* command. Then, only desirable events can be specified in the *perf stat* command with *–e* option.

```
1. echo "gzip -fc -6 /opt/aws/Mdo.seq.all > /dev/null" > cmd.sh
2. ~$ perf stat -x ' ' ./cmd.sh
3.    7841.230460  task-clock
4.    8            context-switches
5.    8            cpu-migrations
6.    211          page-faults
7.    29348483968  cycles
8.    14723162586  stalled-cycles-frontend
9.    10875856918  stalled-cycles-backend
10.   39715515197  instructions
11.   8678562885   branches
12.   260293379    branch-misses
```

Figure 2.12 perf stat output for gzip -6 example

2.4.3 Datasets

Mobile devices. To evaluate the behavior of compression (utility, level) pairs during compressed uploads and downloads initiated on mobile devices a varied collection of datasets that are representative of mobile computing has been compiled. The datasets include text files, lossless images, executables of the most popular Android applications, source files of Android applications, files containing data from wearable health monitoring devices, offline maps and routing files from MAPS.ME application [61], navigation files from OsmAnd application [62] (part of OpenStreetMaps), and language packages for off-line translation from Google Translate application. Selected files types represent the typical data transmissions performed by the mobile devices to and from the cloud (including web and file servers), such as:

- Download of applications, eBooks, and emails;
- Retrieval of web-pages;
- Download and upload of health logs and raw sensor data generated locally or from connected wearable devices or sensors (e.g., paired via Bluetooth);
- Download of maps and translation packages; and
- File system backup and upload of files to the distributed file storage.

Table 2.3 gives a complete list of datasets used, including their types, the number of files in a set, the total size and a description. Files that are compressed by default (e.g., apk) are repackaged into uncompressed archive files (tar). Files range in size from 0.65 KB to 706.8 MB with the average and median being 10.17 MB and 25 MB, respectively.

Table 2.3 Datasets to characterize local (de)compression on mobile devices

Dataset	Type	# of files	Total Size	Description
apk	binary (apk)	277	7.33 GB	Extracted apk files (from Google Play [25] and F-Droid application repositories [40])
apksource	code	59	1.16 GB	Source files of selected apk files
Books	text (txt)	1067	0.56 GB	Collection of Project Gutenberg Works
DNG	image (dng)	67	1.51 GB	Set of lossless DNG images taken with OnePlus One smartphone
HealthSUM	csv/dat	28	75.20 MB	Files containing periodic logs with the average heart rate, breathing, rate, posture, level of physical activity, skin temperature, min/max acceleration, ECG amplitude, ECG noise level, and battery status; reported once every second
HealthWAVE	csv/dat	86	2.90 GB	Files containing Electrocardiogram, Breathing, and Acceleration waveforms; (CSV files and binary DAT files)
Maps	mwm	51	2.63 GB	Offline maps from MAPS.ME application
Maps_routing	mwm.routing	51	2.46 GB	Offline routing from MAPS.ME application
OsmAnd	srtm.obf	50	7.63 GB	Offline navigation files from OsmAnd mapping application (OpenStreetMaps)
Translate	tar	50	9.41 GB	Offline translation files for all languages from Google Translate application

Workstations. To evaluate the behavior of compression (utility, level) pairs during compressed uploads and downloads initiated on workstations, a varied collection of datasets representative of data transfers to and from the cloud has been compiled. The datasets consist of text files containing DNA sequence data from the UniGene [63] project, binary files containing Earth's surface relief data stored in

NetCDF format collected by the National Oceanic and Atmospheric Administration (NOAA) [64], and XML files containing web data with human-readable English text collected from archived pages on Wikipedia [65]. Selected files represent the typical examples in the areas of scientific and big data processing when data is transferred between the local workstation and the cloud or distributed centers, including:

- Big data text analytics: Analysis of data from web source (tweets, emails, web clickstream, social message);
- Use of detailed relief data in research and simulations involving area of navigation, rescue, and weather predictions and analytics; and finally,
- Research and analysis in genome studies between different collaboration parties with or without cloud offloading;

Table 2.4 gives a complete list of the files in three selected datasets, including dataset name and its selected subset, their types, number of files, total size, and a description. Files range in size from 1.61 MB to 1.87 GB with the average and median being 592.36 MB and 333.29 MB for *wikipages* files, 263.77 MB and 157.38 MB for *netcdf* files, and 216.5 MB and 121.76 MB for *seq.all* files.

Table 2.4 Datasets to characterize local (de)compression on the workstation

Datasets [subset name]	**Type**	**# of files**	**Total Size (GB)**	**Description**
EnWiki [wikipages]	web (xml)	58	90.3	Collection of archived pages from Wikipedia
NOAA [netCDF]	binary	83	21.4	Earth's surface Relief data stored in NetCDF format
UniGene [seq.all]	genome (txt)	140	29.6	Collection of DNA sequence data stored in text

2.5 Metrics and Experiments

This section describes measured and derived metrics to describe performance and energy efficiency (2.5.1), and a set of experiments conducted to quantify performance and energy metrics (2.5.2).

2.5.1 Metrics

Table 2.5 and Table 2.6 summarize the performance and energy metrics used as well as their definitions.

Compression ratio. We use the compression ratio to evaluate the compression effectiveness of an individual utility and its levels of compression. The compression ratio CR is calculated as the size of the uncompressed input file (US) divided by the size of the compressed file (CS), $CR = US/CS$.

Performance. To evaluate the performance of individual compression utilities and compression levels, we measure the time to compress the uncompressed input file ($T.C$) and the time to decompress ($T.D$) a compressed file generated by that utility with the selected compression level. Instead of using the execution times directly, we use the (de)compression throughput ($Th.C$ [$Th.D$]) expressed in megabytes per second ($Th.C = US/T.C$ [$Th.D = US/T.D$]).

To evaluate the performance of uncompressed data transfer, we measure the time to perform uncompressed upload and download ($T.UUP$ [$T.UDW$]). Throughputs for uncompressed data upload and download ($Th.UUP$ [$Th.UDW$]) are expressed in megabytes per second ($T.UUP = US/T.UUP$ [$T.UDW = US/T.UDW$]). The time to set up a network connection is expressed as $T.SC$. The time for uncompressed upload and download without network connection setup time, $T.UP$ [$T.DW$], convert to the network throughput ($Th.UP$ [$Th.DW$]) when expressed in megabytes

per second (*Th.UP = US/T.UP [Th.DW = US/T.DW]*). To evaluate the performance of compressed transfer, we measure the time to perform compressed data upload and download (*T.CUP [T.CDW]*). Throughputs for compressed data upload and download (*Th.CUP [Th.CDW]*) are expressed in megabytes per second (*Th.CUP = US/T.CUP [Th.CDW = US/T.CDW]*).

The derived throughputs capture the efficiency of data transfers from the user's point of view – users produce and consume uncompressed data and care more about the time it takes to transfer data than about what approach is used internally to make the transfer fast. In addition, this metric is suitable for evaluating networked data transfers and comparing compressed and uncompressed transfers.

Table 2.5 Metrics: Performance

Symbol	Description	Units	Definition
US	Uncompressed file size	MB	Measured
CS	Compressed file size	MB	Measured
CR	Compression ratio	-	US/CS
$T.C$ [$T.D$]	Time for local [de]compress	s	Measured
$T.UUP$ [$T.UDW$]	Time for uncompressed upload [download]	s	Estimated
$T.SC$	Time to set up network connection	s	Estimated
$T.UP$ [$T.DW$]	Time for uncompressed upload [download] w/o network connection time	s	$T.UP = T.UUP - T.SC$ [$T.DW = T.UDW - T.SC$]
$T.CUP$ [$T.CDW$]	Time for compressed upload [download]	s	Estimated
$Th.C$ [$Th.D$]	Local [De]compression throughput	MB/s	$US/T.C$ [$US/T.D$]
$Th.UUP$ [$Th.UDW$]	Uncompressed upload [download] throughput	MB/s	$US/T.UUP$ [$US/T.UDW$]
$Th.UP$[$Th.DW$]	Network throughput	MBs	Estimated
$Th.CUP$ [$Th.CDW$]	Compressed upload [download] throughput	MB/s	$US/T.CUP$ [$US/T.CDW$]

Energy efficiency. For each compression task with a selected compression level, we calculate the total energy for compression (*ET.C*) as well as the total energy for decompression (*ET.D*). Instead of reporting the energy directly in joules, we report the energy efficiency (*EE.C [EE.D]*) in megabytes per joule (*EE.C = US/ET.C [EE.D = US/ET.D]*). To eliminate the effects of idle current, we also consider the overhead energies *ET.C(0)* and *ET.D(0)* and derived energy efficiency metrics *EE.C(0)* and *EE.D(0).*

For each uncompressed data transfer, we calculate the total upload energy ($ET.UUP$) as well as the total download energy ($ET.UDW$). Instead of reporting the energy directly in joules, we report the energy efficiency ($EE.UUP$ [$EE.UDW$]) in megabytes per joule ($EE.UUP = US/ET.UUP$ [$EE.UDW = US/ET.UDW$]). The total energy to set up a network connection is expressed as *ET.SC*. The total energy for uncompressed upload and download without energy for network connection setup time, *ET.UP [ET.DW]*, convert to network energy efficiency (*EE.UP [EE.DW]*) when expressed in megabytes per second ($EE.UP$ [$EE.DW$]) when expressed in megabytes per second. For compressed data transfer, we calculate the total upload energy ($ET.CUP$) as well as the total download energy ($ET.CDW$). Instead of reporting the energy directly in joules, we report the energy efficiencies ($EE.CUP$ [$EE.CDW$]) in megabytes per joule ($EE.CUP = US/ET.UUP$ [$EE.CDW = US/ET.UDW$]).

Table 2.6 Metrics: Energy

Symbol	Description	Units	Definition
$ET.C[ET.D]$	Total energy for local [de]compression	J	Estimated
$ET.C(0)$ $[ET.D(0)]$	Overhead energy for [de]compression	J	$ET.C - I_{idle} \cdot V_{bs} \cdot T.C$ $[ET.D - I_{idle} \cdot V_{bs} \cdot T.D]$
$ET.SC$	Total energy to set up network connection	J	Estimated
$ET.UUP$ $[UDW]$	Total energy for uncompressed upload [download]	J	Estimated
$ET.UP[ET.DW]$	Total energy for uncompressed upload [download] w/o energy for network connection time	s	$ET.UP = ET.UUP - ET.SC$ $[ET.DW = ET.UDW - ET.SC]$
$ET.CUP$ $[ET.CUP]$	Total energy for compressed upload [download]	J	Estimated
$EE.C$ $[EE.D]$	Local [de]compression energy efficiency	MB/J	$US/ET.C$ $[US/ET.D]$
$EE.C(0)$ $[EE.D(0)]$	Local [de]compression overhead energy efficiency	MB/J	$US/ET.C(0)$ $[US/ET.D(0)]$
$EE.UUP$ $[EE.UDW]$	Uncompressed upload[download] energy efficiency	MB/J	$US/ET.UUP$ $[US/ET.UDW]$
$EE.UP$ $[EE.DW]$	Network throughput energy efficiency	MB/J	Estimated
$EE.CUP$ $[EE.CDW]$	Compressed upload[download] energy efficiency	MB/J	$US/ET.CUP$ $[US/ET.CDW]$

2.5.2 Experiments

To evaluate the throughput and energy efficiency of (de)compression tasks, we consider set of scenarios for cases when a mobile device and a workstation are used as the target platforms.

For the case when a mobile device is used as the target platform, we consider two typical usage scenarios as illustrated in Figure 2.13. The first experiment is performed locally on the mobile device (LOCAL) involving measuring of time and energy of (de)compression tasks. To eliminate latencies and energy overheads caused by writing files to the internal storage, the output of the (de)compression tasks is re-

directed to the null device (/dev/null) – a special 'file' that discards all data written to it. This experiment is used for each file in the datasets to generate historical prediction tables discussed in the later sections on framework design.

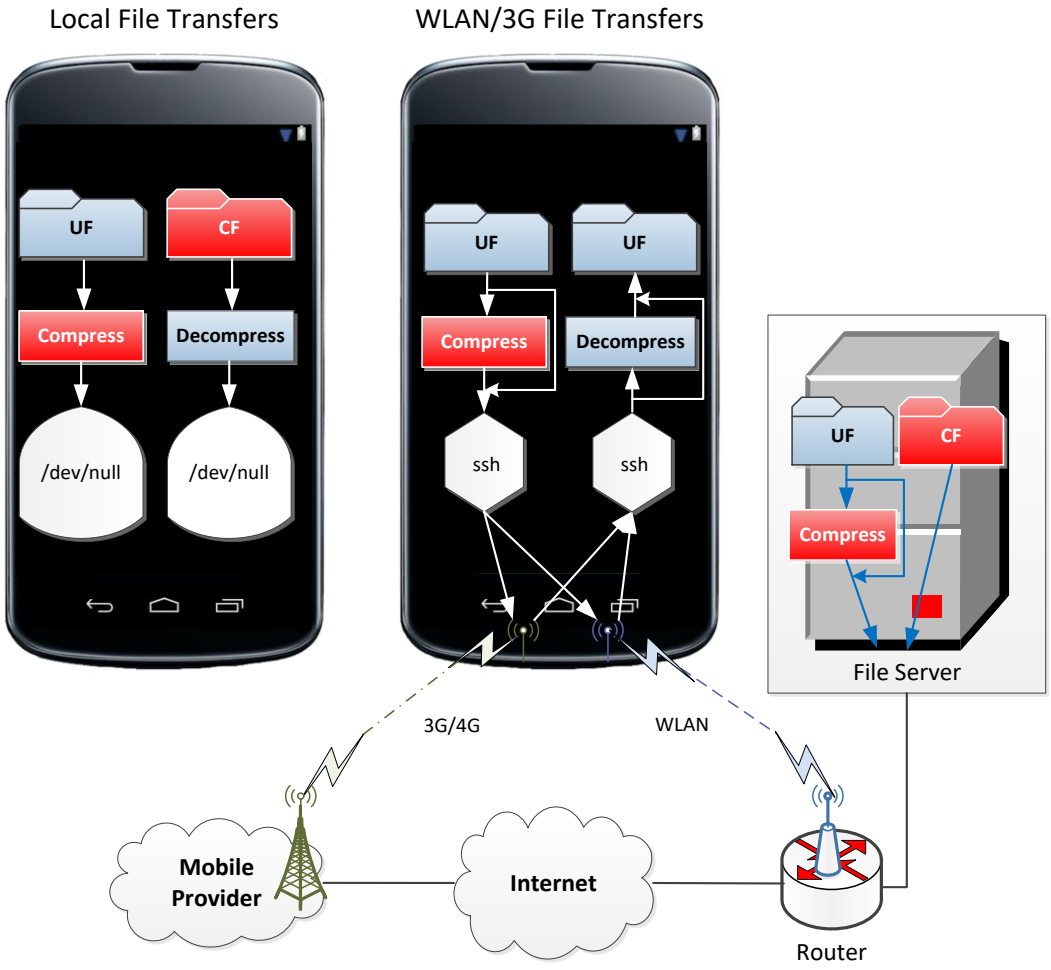


Figure 2.13 Data flow of the experiments on mobile devices (blue file icons refer to uncompressed files, red file icons refer to compressed files).

The second group of experiments (WLAN and CELL) involves measuring the time and energy of compressed uploads and downloads to/from a remote server. For

compressed uploads, the uncompressed input file (UF) is read from the local file system, compressed on the mobile device, and streamed to the server over a secure channel. The output files are redirected to the null device of the server. For compressed downloads, when the compressed files (CF) are maintained on the server, the compressed file is retrieved from the server's file system through a secure channel and decompressed on the mobile device. When compressed files are not maintained on the server, the uncompressed input file is first compressed on the server and then streamed to the mobile device over a secure channel for decompression. In both cases, the output files are redirected to the null device of the smartphone. Those types of experiments are be used to evaluate the effectiveness of the proposed framework.

For the case when a workstation is used as the target platform, we consider two typical usage scenarios as illustrated in Figure 2.14. The first experiment is performed locally on the workstation (LOCAL) involving measuring of time and energy of (de)compression tasks. Likewise, to eliminate latencies caused by writing files to the internal storage, the output of the (de)compression tasks is re-directed to the null device (/dev/null). This experiment is used for each file in the datasets to generate historical prediction tables discussed in the later sections on framework design.

The second group of experiments (LAN) involves measuring the time of compressed uploads and downloads to/from a remote instance on the Amazon's AWS EC2 cloud. For compressed uploads, the uncompressed input file (UF) is read from the local file system, compressed on the workstation, and streamed to the cloud instance over a secure LAN channel. The output files are redirected to the null device of the cloud. For compressed downloads, when the compressed files (CF) are maintained in the cloud, the compressed file is retrieved from the cloud's file system

through a secure LAN channel and decompressed on the workstation. When compressed files are not maintained in the cloud, the uncompressed input file is first compressed on the cloud and then streamed to the workstation over a secure LAN channel for decompression. In both cases, the output files are redirected to the null device of the workstation. These types of experiments are used to evaluate the effectiveness of the proposed framework.

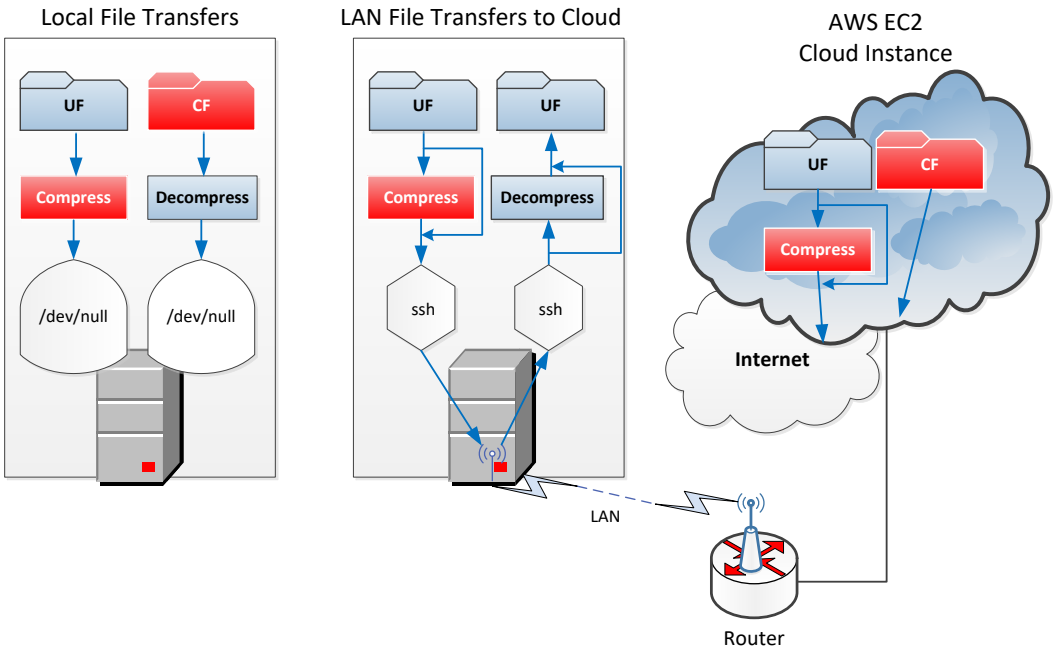


Figure 2.14 Data flow of the experiments on workstation (blue file icons refer to uncompressed files, red file icons refer to compressed files).

2.6 The Cases for Intelligent File Transfers

This section gives two studies and makes the case for intelligent file transfers. The studies analyze both throughput and energy efficiency of uncompressed

and compressed transfer modes on the smartphone (Section 2.6.1) and the workstation (Section 2.6.2).

2.6.1 Compressed Transfers on the Smartphone

Smartphone users upload and download a diverse set of data. As a data upload examples, we consider uploading mHealth data files to the server that include physiological information (e.g., breathing waveform or health summary log). As a data download examples, we consider downloading applications and additional data such as book, source code, and map file from data repository on the smartphone. Applications running on smartphones and users have several options on how they can perform an upload or download of any particular file. Files can be transferred uncompressed, by using the default compression utility (usually a variant of *gzip* -6 or *zip* compression), or by selecting another compression utility and compression level. In this case study, we show that a compression utility and a compression level that achieves the maximum throughput and energy efficiency changes as a function of network conditions and file parameters, such as file size and type.

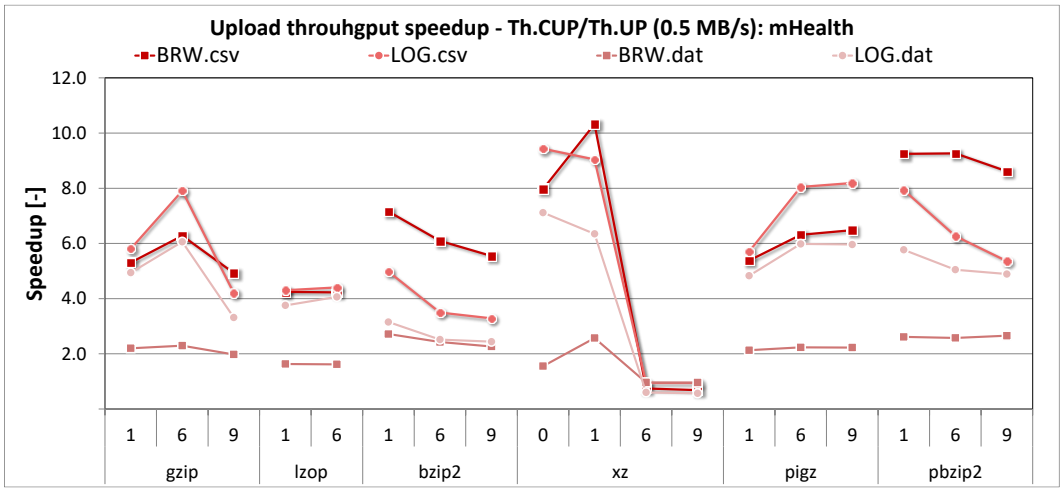
2.6.1.1 Upload Examples

We consider uploading two text and two binary files from the smartphone. The first file (BRW.csv) contains breathing data recorded by a wearable health monitor - Zephyr Technologies BioHarness 3. The file contains raw samples from monitor's breathing sensor during a subject's 6-hour sleep. The breathing sensor is sampled with the frequency of 100 Hz. The second file (LOG.csv) contains a log of user's physiological state captured by the same monitor during activities of daily living that include walking, driving, and office work. The log is taken every second and in-

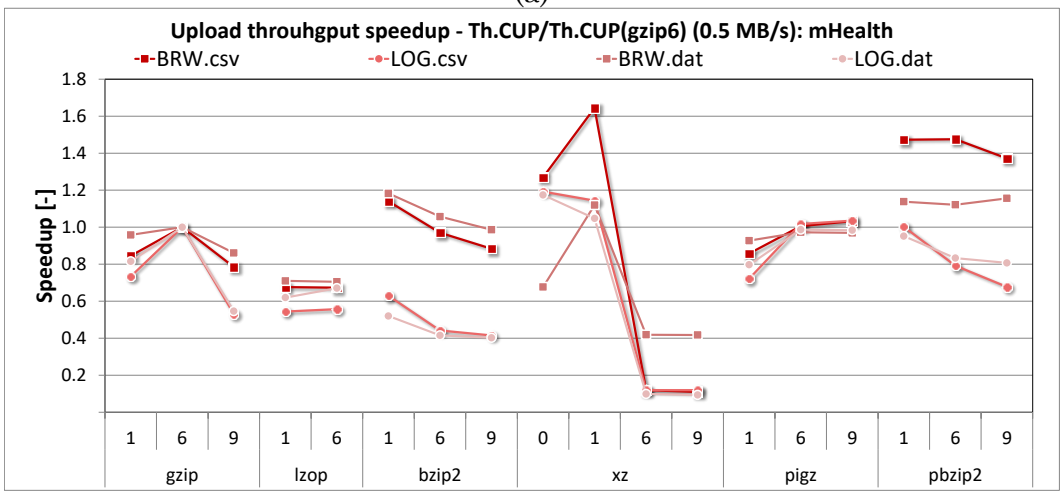
cludes information about heart rate, breathing rate, and a level of physical activity. The files BRW.dat and LOG.dat contain the same information as BRW.csv and LOG.csv, but in binary *.dat* format, custom for the selected wearable health monitor. These types of mobile health data are often uploaded to the cloud where more sophisticated processing can take place. For example, we can extract the subject's type and level of physical activity during the day, or analyze the quality of sleep during the night. The uncompressed file sizes for the text files are 19.75 MB for BRW.csv and 4.69 MB for LOG.csv, whereas the binary file sizes are 2.39 MB for BRW.dat and 3.26 MB for LOG.dat. The experiment involves uncompressed and compressed file uploads. For each transfer mode, the total time and energy spent to upload a file are measured to determine the effective upload throughput and energy efficiency.

Upload throughput. Figure 2.15 shows the effective upload throughput speedups for all compressed upload modes relative to the corresponding uncompressed transfers (a) and the default compressed transfers with *gzip* -6 (b) when using the 0.5 MB/s WLAN network. The effective throughputs of uncompressed uploads match the network throughput for all four files. The default compressed uploads with *gzip* -6 achieve the effective throughputs of 3.31 MB/s and 4.05 MB/s for BRW.csv and LOG.csv, and 1.14 MB/s and 3.08 MB/s for binary BRW.dat and LOG.dat. The utilities with the highest effective upload throughputs are *bzip2* and *xz* because they achieve relatively high compression ratios. For BRW.csv, the best effective throughput is achieved by *xz* -1, offering 10.33- and 1.64-fold improvements in the effective throughput relative to the uncompressed upload and the default compressed upload, respectively. For LOG.csv, the best effective throughput is achieved by *xz* -0 (9.43- and 1.19-fold improvements). For BRW.dat, the best effective

throughput is achieved by *bzip2* -1 (2.72- and 1.18-fold improvements). Finally, for LOG.dat, the best effective throughput is achieved by *xz* -0 (7.12- and 1.17-fold improvements).



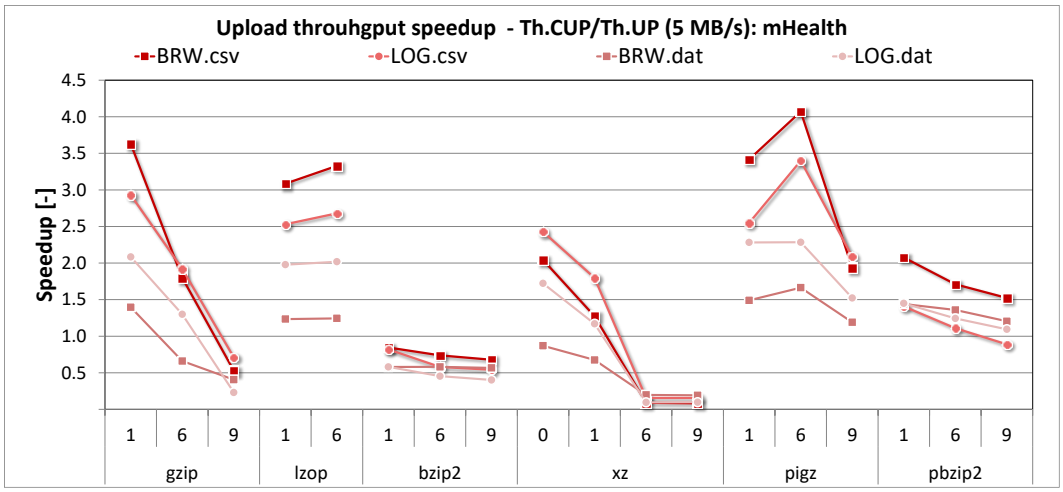
(a)



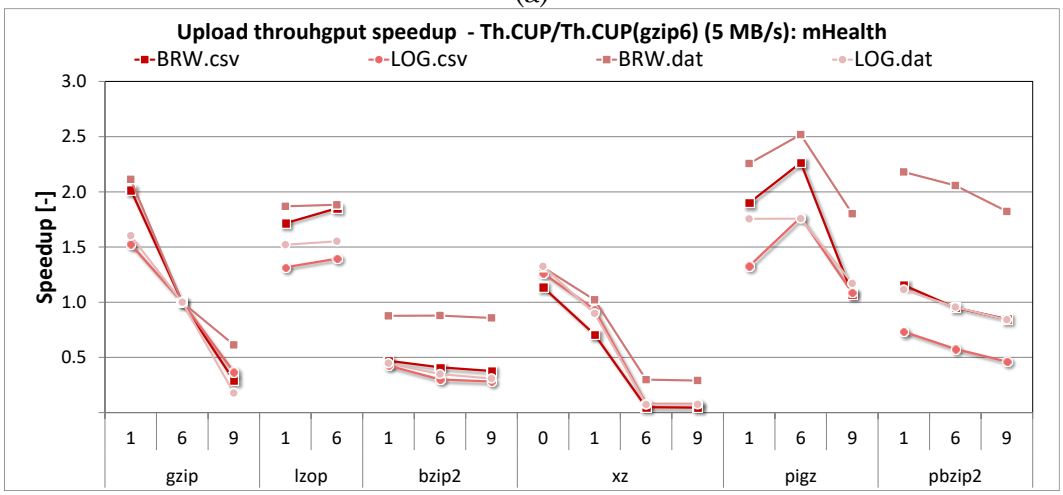
(b)

Figure 2.15 Upload throughput speedup for mHealth files on 0.5 MB/s WLAN connection

Figure 2.16 shows the effective upload throughput speedups for all compressed upload modes relative to the corresponding uncompressed transfers (a) and the default compressed transfers with *gzip* -6 (b) when using the 5 MB/s WLAN network. The uncompressed uploads achieve the effective throughputs of 4.43 MB/s and 3.16 MB/s for the text files, and 2.52 MB/s and 3.07 MB/s for the binary files. Please note that the effective throughput of uncompressed uploads directly depends on the uncompressed file size – it is higher for larger files (e.g., 19.75 MB BRW.csv file) and lower for smaller files (e.g., 2.39 MB BRW.dat file). The default compressed upload with *gzip* -6 achieves the effective throughputs of 7.96 MB/s and 6.07 MB/s for the text files, and 1.66 MB/s and 3.99 MB/s for the binary files. The utility with the highest effective upload throughputs is *pigz*. For BRW.csv, the best effective throughput is achieved by *pigz* -6, offering 4.07- and 2.27-fold improvements in the effective throughput relative to the uncompressed upload and the default compressed upload, respectively. For LOG.csv, the best effective throughput is achieved by *pigz* -6 (3.4- and 1.77-fold improvements). For BRW.dat, the best effective throughput is achieved by *pigz* -6 (1.66- and 2.52-fold improvements). Finally, for LOG.dat, the best effective throughput is achieved by *pigz* -6 (2.28- and 1.76-fold improvements).



(a)

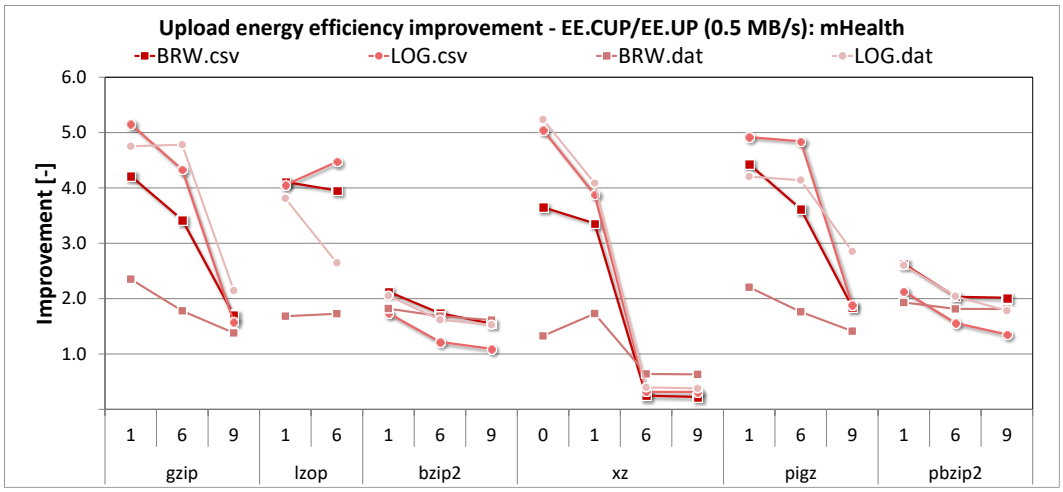


(b)

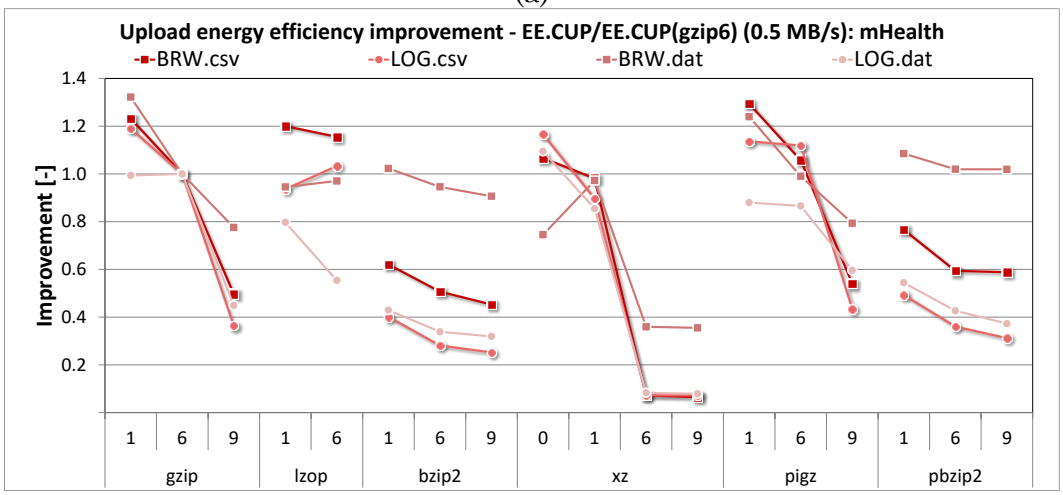
Figure 2.16 Upload throughput speedup for mHealth files on 5 MB/s WLAN connection

Upload energy efficiency. Figure 2.17 shows the effective upload energy efficiency improvements for all compressed upload modes relative to the corresponding uncompressed transfers (a) and the default compressed transfers with *gzip* -6 (b) when using the 0.5 MB/s WLAN network. The uncompressed uploads achieve the effective energy efficiencies of 0.92 MB/J and 0.88 MB/J for the text files, and 0.63 MB/J and 0.7 MB/J for the binary files. The default compressed upload with

gzip -6 achieves the effective energy efficiencies of 3.16 MB/J and 3.82 MB/J for the text files, and 1.11 MB/J and 3.34 MB/J for the binary files. The utilities with the highest effective upload energy efficiency are *pigz* -1, *gzip* -1 and *xz* -0. For BRW.csv, the best effective energy efficiency is achieved by *pigz* -1, offering 4.43- and 1.29-fold improvements in energy efficiency over the uncompressed upload and the default compressed upload, respectively. For LOG.csv, the best effective energy efficiency is achieved by *gzip* -1 (5.15- and 1.17-fold improvements). For BRW.dat, the best effective energy efficiency is achieved by *gzip* -1 (2.35- and 1.32-fold improvements). For LOG.dat, the best effective energy efficiency is achieved by *xz* -0 (5.23- and 1.09-fold improvements).



(a)

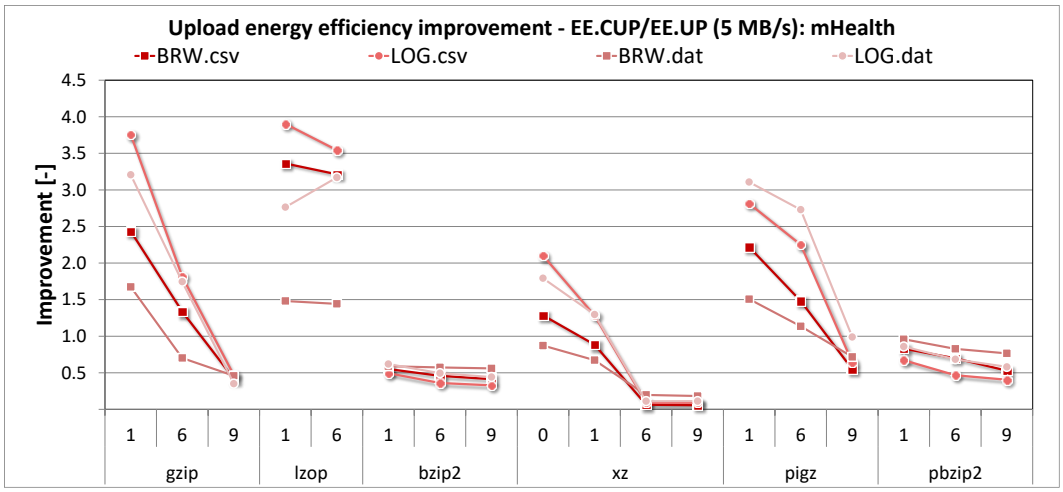


(b)

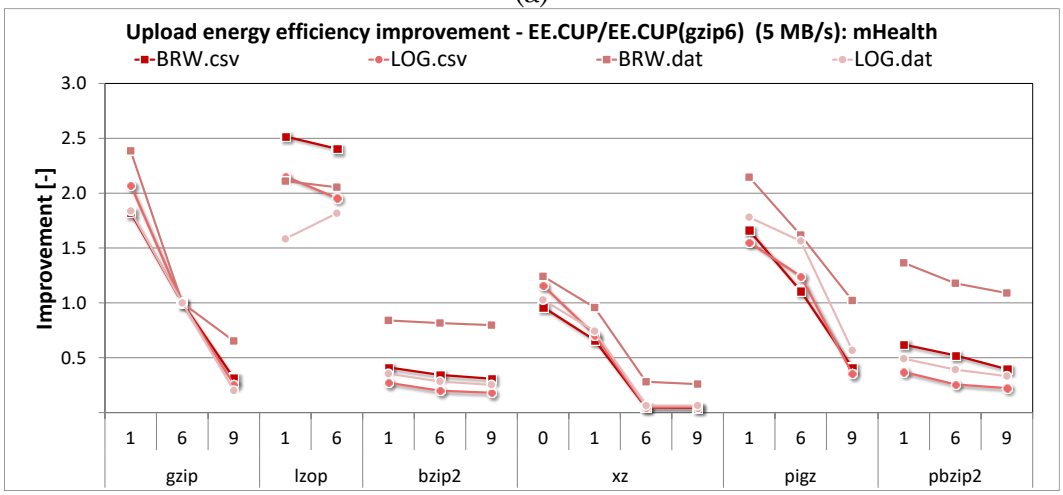
Figure 2.17 Upload energy efficiency improvement for mHealth files on 0.5 MB/s WLAN connection

Figure 2.18 shows the effective upload energy efficiency improvements for all compressed upload modes relative to the corresponding uncompressed transfers (a) and the default compressed transfers with *gzip* -6 (b) when using the 5 MB/s WLAN network. The uncompressed uploads achieve the effective energy efficiencies of 3.6 MB/J and 2.99 MB/J for the text files, and 2.09 MB/J and 2.45 MB/J for the binary files. Please note that the effective energy efficiency of the uncompressed upload

directly depends on the uncompressed file size – it is higher for larger files (e.g., 19.75 MB BRW.csv file) and lower for smaller files (e.g., 2.39 MB BRW.dat file). The default compressed upload with *gzip* -6 achieves the effective energy efficiencies of 4.81 MB/J and 5.42 MB/J for the text files, and 1.47 MB/J and 4.28 MB/J for the binary files. The utilities with the highest effective upload throughputs are *lzop -1* and *gzip -1*. For BRW.csv, the best effective energy efficiency is achieved by *lzop* -1, 3.36- and 2.52-fold improvements in the effective throughput relative to the uncompressed upload and the default compressed upload, respectively. For LOG.csv, the best effective energy efficiency is achieved by *lzop* -1 (3.9- and 2.15-fold improvements). For BRW.dat, the best effective energy efficiency is achieved by *gzip* -1 (1.67- and 2.38-fold improvements). For LOG.dat, the best effective energy efficiency is achieved by *gzip* -1 (3.21- and 1.84-fold improvements).



(a)



(b)

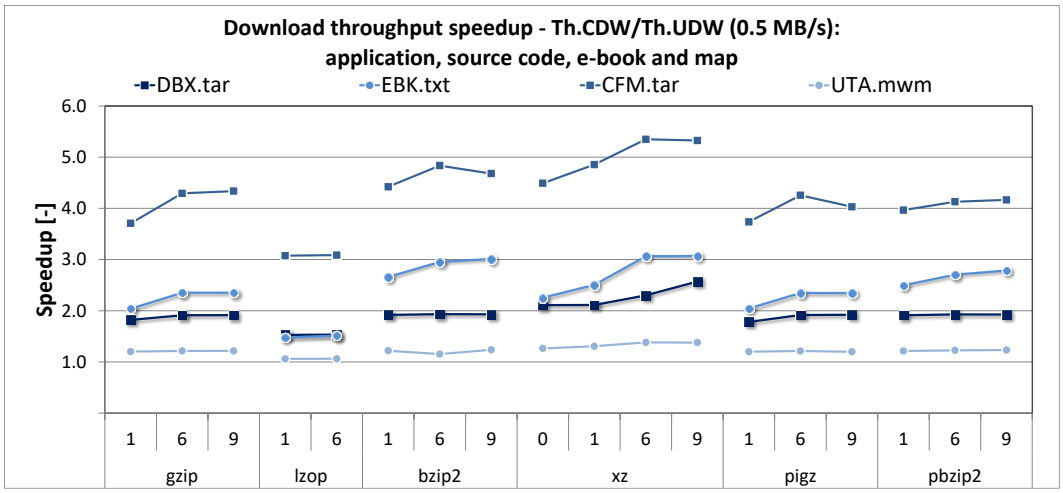
Figure 2.18 Upload energy efficiency improvement for mHealth files on 5 MB/s WLAN connection

2.6.1.2 Download Examples

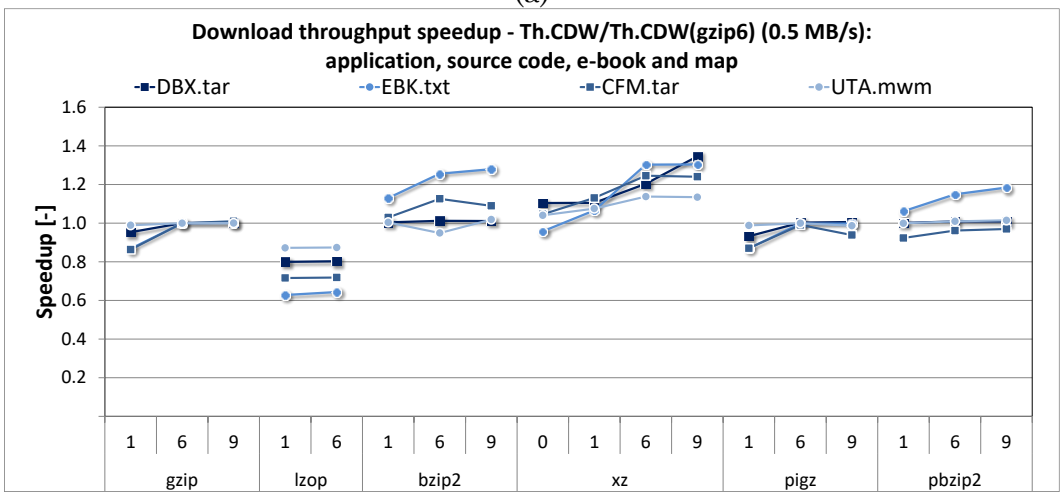
To evaluate the effectiveness of compressed download on the smartphone, we consider uncompressed and compressed downloads of an executable, a text file, a source code, and a map data from the cloud. The first file (DBX.tar) contains an executable for the Android Dropbox application. The second file (EBK.txt) contains an e-book from the Project Gutenberg collection. The third file (CFM.tar) contains a

source code of CyanogenMod File Manager. The fourth file (UTA.mwm) contains an offline map data from Maps.me Android application. The uncompressed file sizes are 69.31 MB for the DBX.tar, 5.44 MB for the EBK.txt, and 4.58 MB and 27.43 MB for the CFM.tar and the UTA.mwm, respectively. The experiment involves uncompressed and compressed file downloads. For compressed file downloads, all compressed versions of the files are made available in the cloud. For each transfer mode, the total time and energy spent to get the uncompressed file are measured to determine the effective download throughput and energy efficiency.

Download throughput. Figure 2.19 shows the effective download throughput speedups for all compressed download modes relative to the corresponding uncompressed transfers (a) and the default compressed transfers with *gzip* -6 (b) when using the 0.5 MB/s WLAN network. The effective throughputs of uncompressed downloads match the network throughput for all four files. The default compressed downloads with *gzip* -6 achieve the effective throughputs of 1.02 MB/s and 1.23 MB/s for DBX.tar and EBK.txt, and 2.22 MB/s and 0.65 MB/s for CFM.tar and UTA.mwm. The utility with the highest effective download throughputs is *xz*. For DBX.tar, the best effective throughput is achieved by *xz* -9, offering 2.57- and 1.35-fold improvements in the effective throughput relative to the uncompressed download and the default compressed download, respectively. For EBK.txt, the best effective throughput is achieved by *xz* -6 and -9 (3.07- and 1.30-fold improvements). For CFM.tar, the best effective throughput is achieved by *xz* -6 (5.35- and 1.25-fold improvements). For UTA.mwm, the best effective throughput is achieved by *xz* -6 (1.38- and 1.14-fold improvements).



(a)

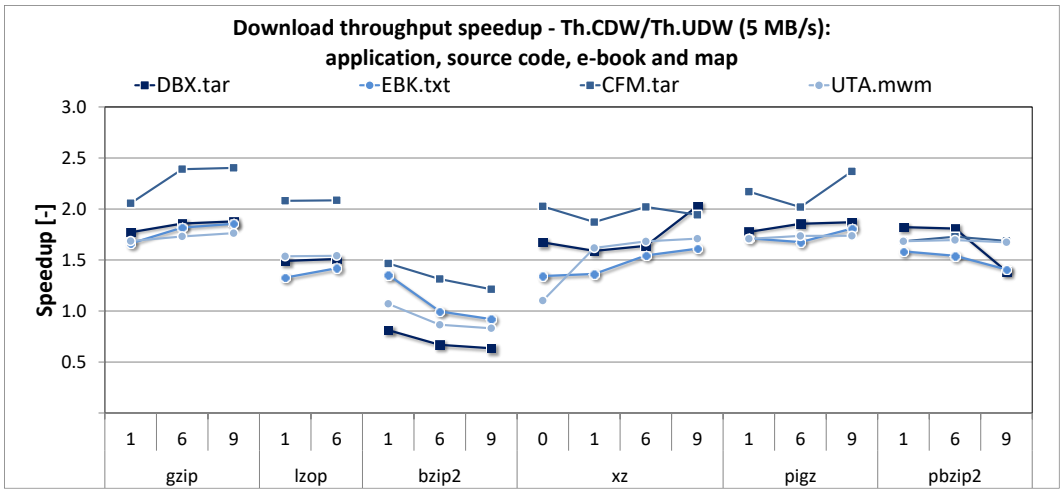


(b)

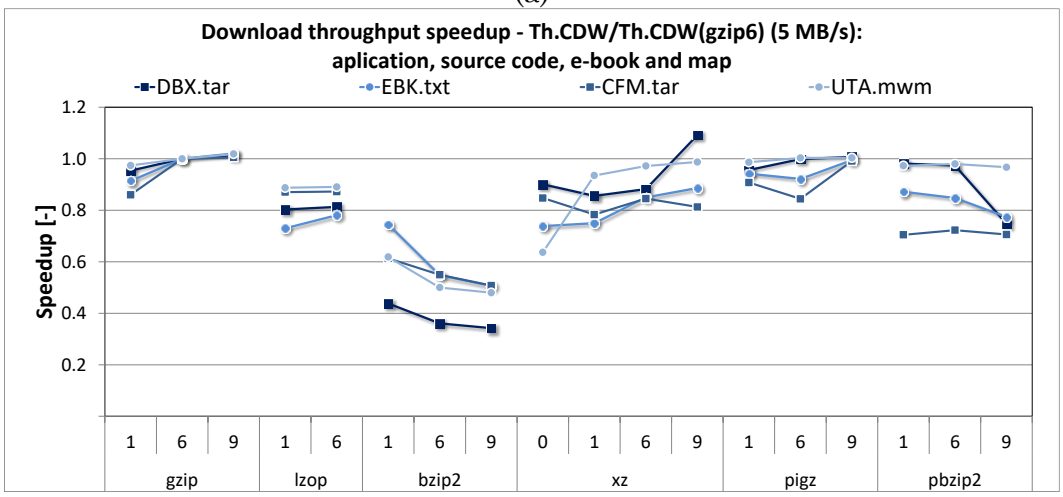
Figure 2.19 Download throughput speedup for application, source code, e-book, and map on 0.5 MB/s WLAN connection

Figure 2.20 shows the effective download throughput speedups for all compressed download modes relative to the corresponding uncompressed transfers (a) and the default compressed transfers with *gzip* -6 (b) when using the 5 MB/s WLAN network. The uncompressed downloads achieve the effective throughputs of 4.99 MB/s for DBX.tar, 3.84 MB/s for EBK.txt, 3.58 MB/s for CFM.tar, and 3.25 MB/s for UTA.mwm. The default compressed downloads with *gzip* -6 achieve

the effective throughputs of 9.27 MB/s and 6.97 MB/s for DBX.tar and EBK.txt, and 8.57 MB/s and 5.63 MB/s for CFM.tar and UTA.mwm. The utilities with the highest effective download throughputs are *xz* -9 and *gzip* -9. For DBX.tar, the best effective throughput is achieved by *xz* -9, offering 2.03- and 1.09-fod improvements in throughputs over the uncompressed download and the default compressed download, respectively. For EBK.txt, the highest effective throughput is achieved by *gzip* -9 (1.82- and 1.02-fold improvements). For CFM.tar, the best effective throughput is achieved by *gzip* -9 (2.40- and 1.01-fold improvements). For UTA.mwm, the best effective throughput is achieved by *gzip* -9 (1.76- and 1.02-fold improvements).



(a)

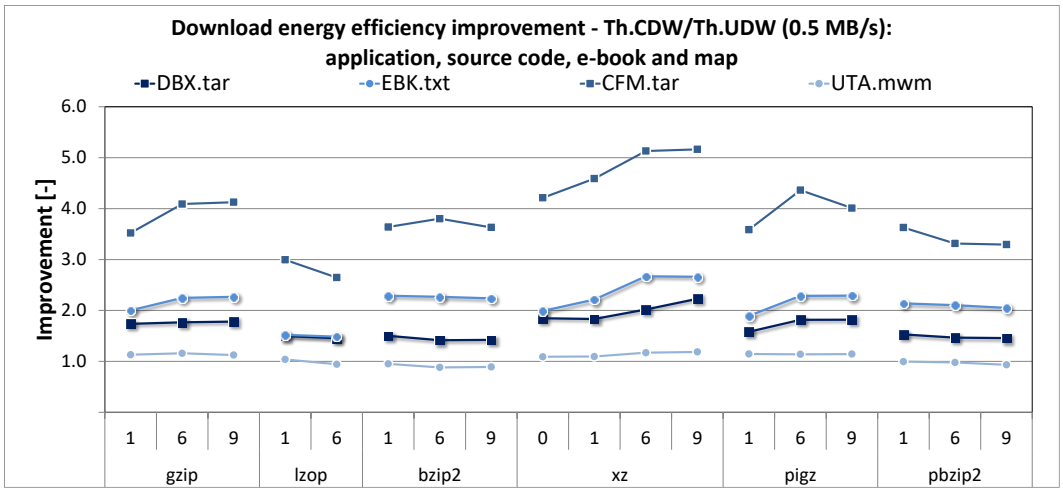


(b)

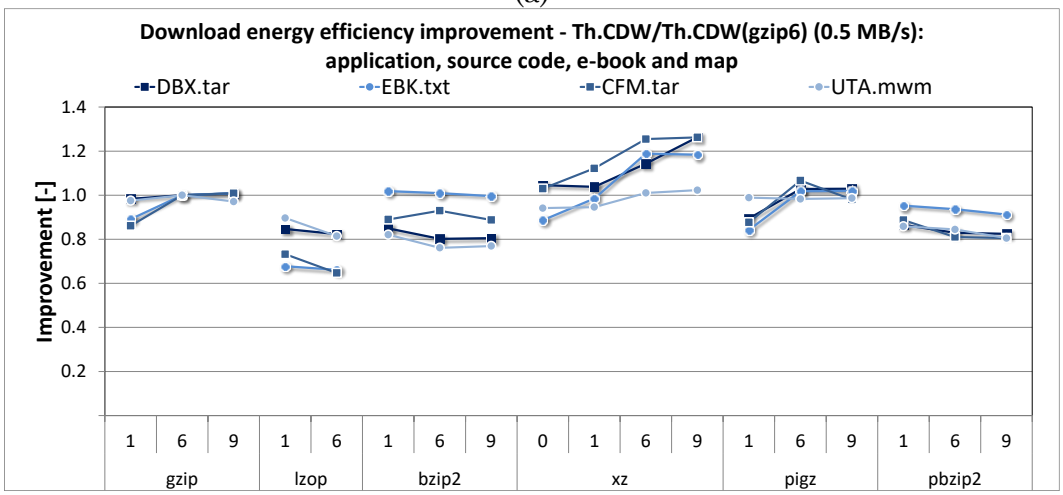
Figure 2.20 Download throughput speedup for application, source code, e-book, and map on 5 MB/s WLAN connection

Download energy efficiency. Figure 2.21 shows the effective download energy efficiency improvements for all compressed download modes relative to the corresponding uncompressed transfers (a) and the default compressed transfers with *gzip* -6 (b) when using the 0.5 MB/s WLAN network. The effective uncompressed download energy efficiencies achieve the effective energy efficiency of 0.9 MB/J for DBX.tar, 0.79 MB/J for EBK.txt, and 0.79 MB/J and 0.89 MB/J for CFM.tar and

UTA.mwm. The default compressed downloads with *gzip* -6 achieve the effective energy efficiencies of 1.59 MB/J and 1.77 MB/J for DBX.tar and EBK.txt, and 3.23 MB/J and 1.03 MB/J for CFM.tar and UTA.mwm. The utility with the highest effective download energy efficiencies is *xz* because it achieves relatively high compression ratios. For DBX.tar, the best effective energy efficiency is achieved by *xz* -9, which results in 2.23- and 1.26-fold improvement in energy efficiency over the uncompressed download and the default compressed download, respectively. For EBK.txt, the best effective energy efficiency is achieved by xz -6 (2.67- and 1.19-fold improvements). For CFM.tar, the best effective energy efficiencies are achieved by *xz* -9 (5.13- and 1.26-fold improvements). For UTA.mwm, the best effective energy efficiencies are achieved by *xz* -9 (1.19- and 1.02-fold improvements).



(a)

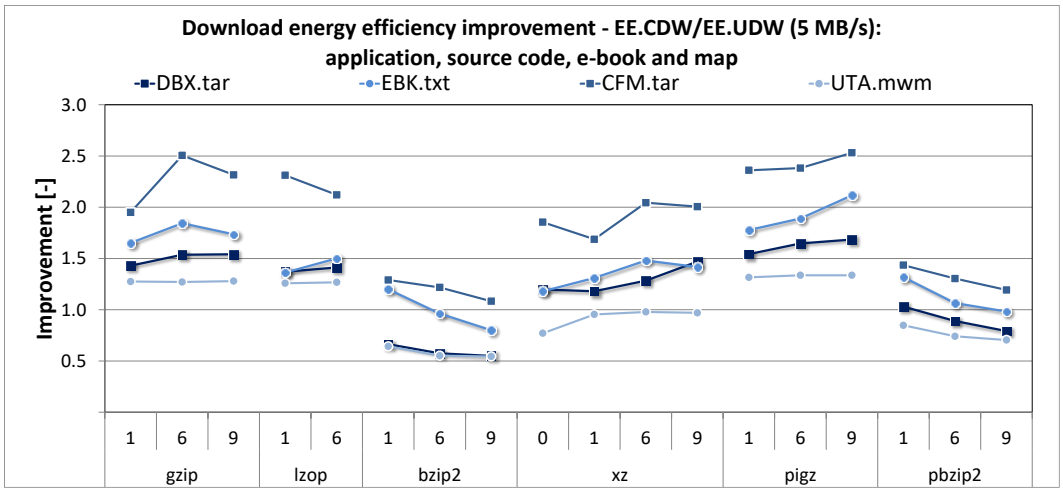


(b)

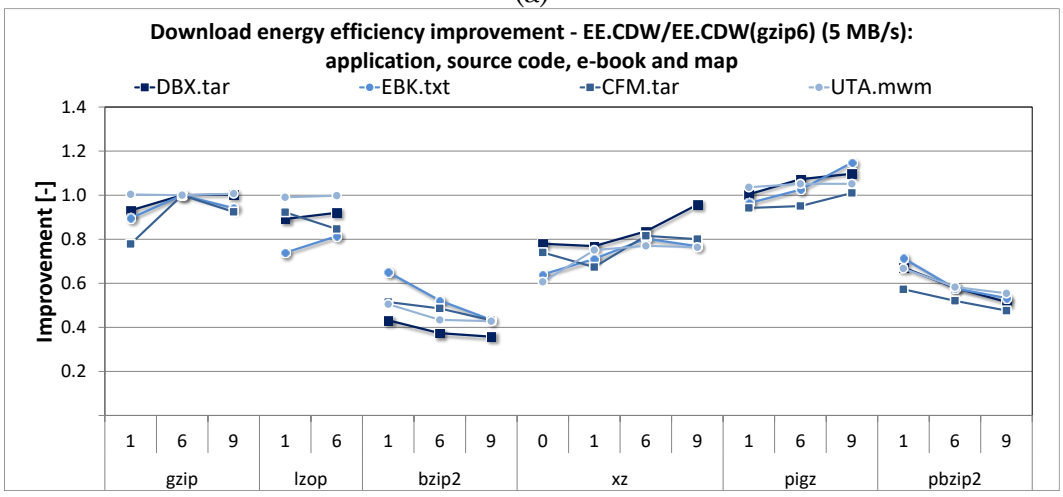
Figure 2.21 Download energy efficiency improvement for application, source code, e-book, and map on 0.5 MB/s WLAN connection

Figure 2.22 shows the effective download energy efficiency improvements for all compressed download modes relative to the corresponding uncompressed transfers (a) and the default compressed transfers with *gzip* -6 (b) when using the 5 MB/s WLAN network. The uncompressed downloads achieve the effective energy efficiencies of 4.28 MB/J for DBX.tar, 3.67 MB/J for EBK.txt, and 3.75 MB/J and 3.26 MB/J for CFM.tar and UTA.mwm. The default compressed downloads with *gzip* -6 achieve

the effective energy efficiencies of 6.58 MB/s and 6.77 MB/s for DBX.tar and EBK.txt, and 9.39 MB/s and 4.14 MB/s for binary CFM.tar and UTA.mwm. The utility with the highest effective download energy efficiencies is *pigz*. For DBX.tar, the best effective energy efficiency is achieved by *pigz* -9, which improves download energy efficiency 1.69- and 1.10-fold relative to the uncompressed and the default compressed downloads. For EBK.txt, the best energy efficiency is achieved by *pigz* -9 (2.83- and 1.15-fold improvement). For CFM.tar, the best energy efficiency is achieved by *pigz* -6 (2.53- and 1.34-fold improvements). For UTA.mwm, the best energy efficiency is achieved by *pigz* -9 (1.34- and 1.05-fold improvements).



(a)



(b)

Figure 2.22 Download energy efficiency improvement for application, source code, e-book, and map on 5 MB/s WLAN connection

2.6.2 Compressed Transfers on Workstations

Workstations used for big data and scientific data analysis can use globally distributed cloud instances or collaborating centers for computational and storage offload. As a data upload example, a local workstation may upload locally generated input files for further processing in the cloud or at collaborating center with more computational power. As a data download example, a local workstation may down-

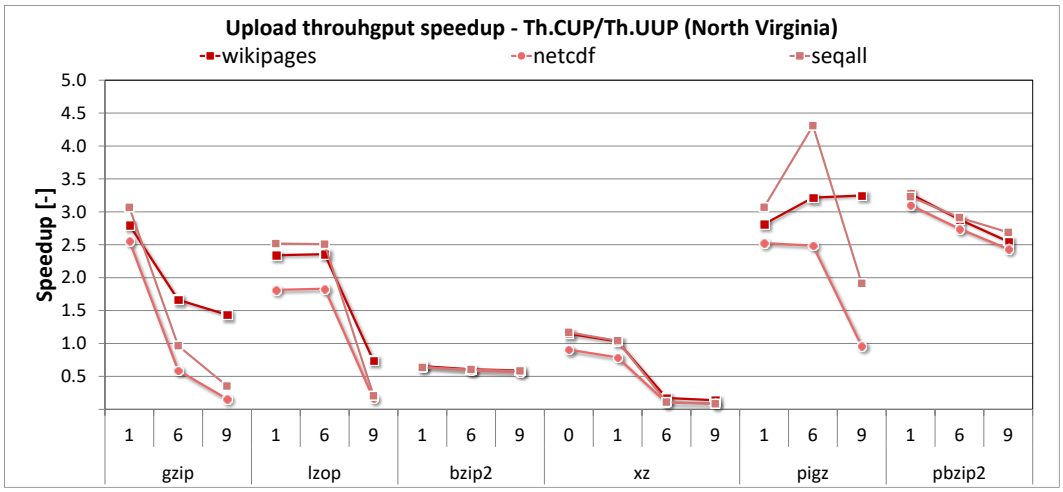
load processed data from the cloud. As with the smartphone case study, the workstation has several options on how to perform an upload or download of any particular file. It can be done without the utilization of any compression by transferring data in an uncompressed format, or with the use of a default compression utility (usually a variant of *gzip* -6 or *zip* compression), or by selecting another compression utility and compression level to perform compressed data transfer. In this case study, we show that a compression utility and a compression level that achieves the maximum throughput and energy efficiency changes as a function of network conditions, dependent on the location of the cloud instance, and file parameters, such as file size and type.

2.6.2.1 Upload Examples

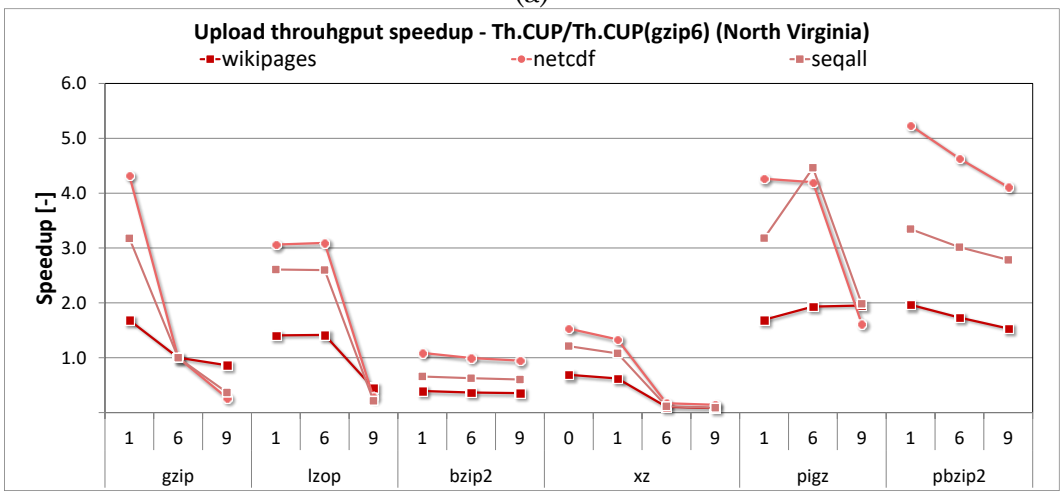
We consider uploading three files (XML, binary, and text) from the workstation to the cloud instance. The first file (wikipages) is an XML file containing web data from archived pages on Wikipedia, the second file (netcdf) is a binary file in NetCDF format containing Earth's surface relief data from the north-eastern USA, and the last file (seqall) is a text file containing DNA sequence of an apple tree, *Malus Domestica*. These types of data are often uploaded to the cloud where more sophisticated processing can take place. For example, we can upload archived pages to extract needed information in the cloud, or to distribute the Earth's surface relief data or DNA sequence files for use across a network of collaborating centers. The uncompressed file sizes for selected files range from 147.6 to 203.7 MB. For wikipages, the uncompressed file size is 147.6 MB, for netcdf, the uncompressed file size is 166.9 MB, and for seqall, it is 203.7 MB. The experiment involves uncompressed and compressed file uploads to the cloud instances located in North Virginia and To-

kyo. For each transfer mode, the total time and energy spent to upload a file are measured to determine the effective upload throughput and energy efficiency. Energy measurements are performed using *likwid-powermeter* from *likwid-tools* [49], [50], which is used to capture energy consumed within CPU package (PKG).

Upload throughput. Figure 2.23 shows the effective upload throughput speedups for all compressed upload modes relative to the corresponding uncompressed transfers (a) and the default compressed transfers with *gzip* -6 (b) to the North Virginia instance when using the uncapped LAN network. The effective throughputs of uncompressed uploads match the network throughput for all three files, being ~ 17.9 M B/s. The default compressed uploads with *gzip* -6 achieve the effective throughputs of 29.25 MB/s for wikipages, 10.51 MB/s for netcdf, and 17.3 MB/s for seqall. Only the default upload of wikipages outperforms its uncompressed upload. The utilities with the highest effective upload throughputs are *pigz* and lower levels of *pbzip2* because they achieve relatively fast compression with parallelization. For wikipages, the best effective throughput is achieved by *pbzip2* -1, offering 3.27- and 1.97-fold improvements in throughput over the uncompressed upload and the default compressed upload, respectively. For netcdf, the best effective throughput is achieved by *pbzip2* -1 (3.1- and 5.24-fold improvements). For seqall, the best effective throughput is achieved by *pigz* -6 (4.31- and 4.46-fold improvements).



(a)

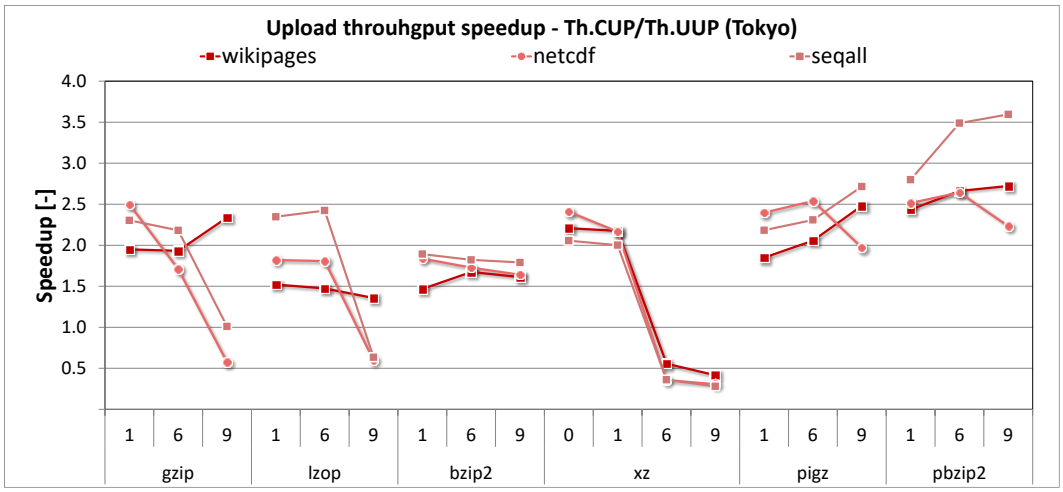


(b)

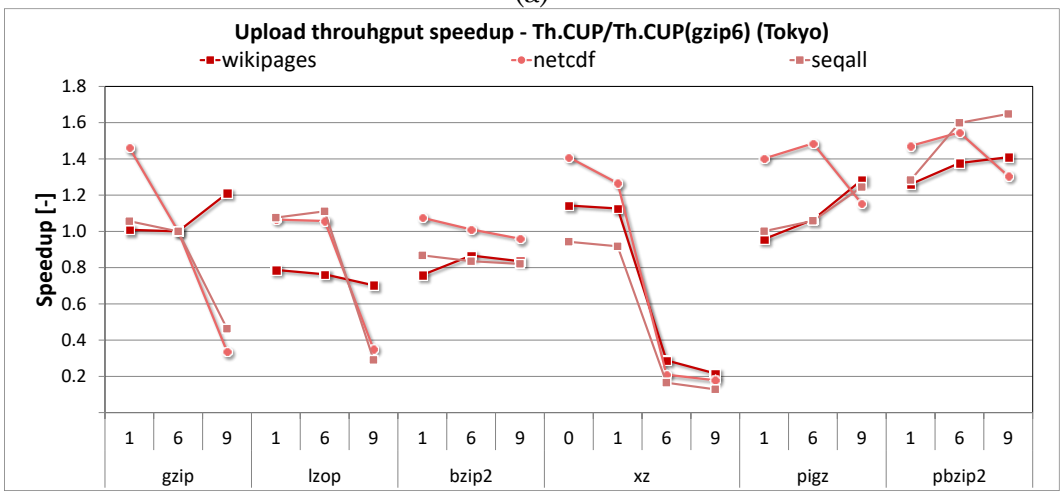
Figure 2.23 Upload throughput speedup for upload of wikipages, netcdf, and seqall files to North Virginia on uncapped LAN connection

Figure 2.24 shows the effective upload throughput speedups for all compressed upload modes relative to the corresponding uncompressed transfers (a) and the default compressed transfers with *gzip* -6 (b) to the Tokyo instance when using the uncapped LAN network. The uncompressed uploads achieve the effective throughputs of 5.25 MB/s for wikipages, 4.75 MB/s for netcdf, and 5.14 MB/s for seqall. Please note that the effective throughput of the uncompressed upload directly

depends on the location of cloud instance – highest for North Virginia (with higher network throughput and lower time to set up a network connection), and lowest for Tokyo (with lower network throughput and higher time to set up a network connection). The default compressed uploads with *gzip* -6 achieve the effective throughputs of 10.15 MB/s for wikipages, 8.12 MB/s for netcdf, and 11.2 MB/s for seqall. The utility with the highest effective upload throughputs is *pbzip2* because it achieves relatively high compression ratios and fast compression with use of parallelization. For wikipages, the best effective throughput is achieved by *pbzip2* -9, offering 2.72- and 1.41-fold improvements in the effective throughput over the uncompressed upload and the default compressed upload, respectively. For netcdf, the best effective throughput is achieved by *pbzip2* -6 (2.64- and 1.55-fold improvements). For seqall, the best effective throughput is achieved by *pbzip* -9 (3.6- and 1.65-fold improvements).



(a)

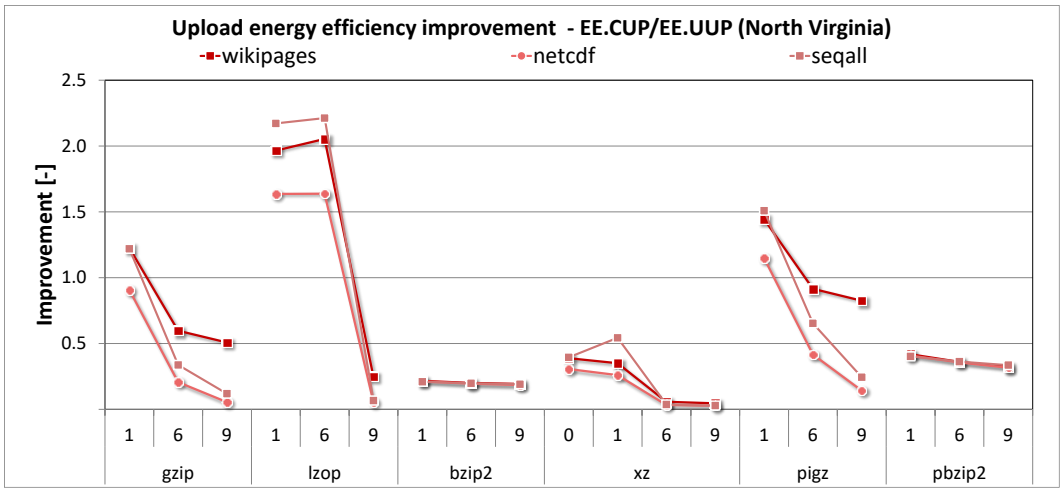


(b)

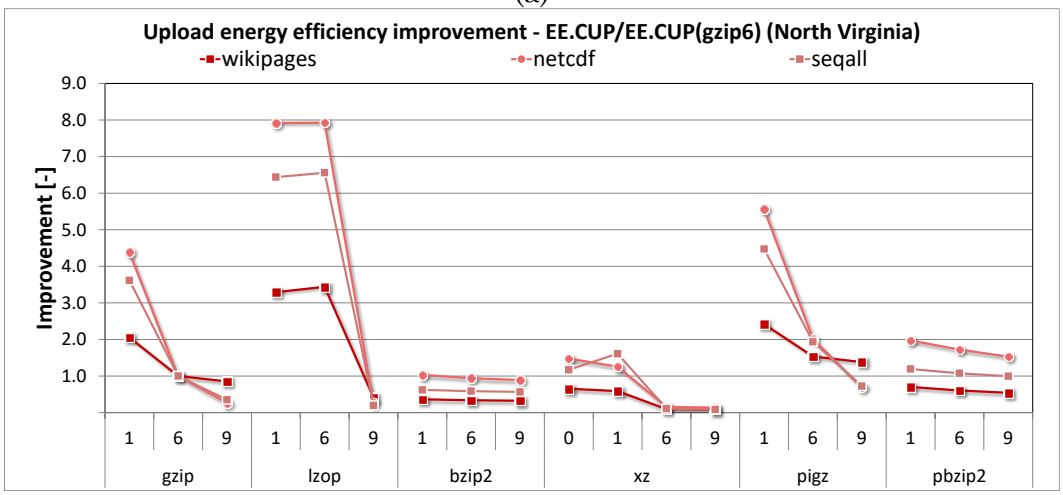
Figure 2.24 Upload throughput speedup for upload of wikipages, netcdf, and seqall files to Tokyo on uncapped LAN connection

Upload energy efficiency. Figure 2.25 shows the effective upload energy efficiency improvements for all compressed upload modes relative to the corresponding uncompressed transfers (a) and the default compressed transfers with *gzip* -6 (b) to the North Virginia instance when using the uncapped LAN network. The uncompressed uploads achieve the effective energy efficiencies ranging from 3.21 MB/J to 3.25 MB/J for all three files. The default compressed upload with *gzip* -6 achieves

the effective energy efficiencies of 1.92 MB/J for wikipages, 0.66 MB/J for netcdf, and 1.1 MB/J for seqall, all of which are below energy efficiency of uncompressed transfers. The utility with the highest effective upload energy efficiencies is *lzop*. For wikipages, the best effective energy efficiency is achieved by *lzop* -6, offering 2.05- and 3.44-fold improvements in the effective energy efficiency over the uncompressed upload and the default compressed upload, respectively. For netcdf, the best effective energy efficiency is achieved by *lzop* -6 (1.64- and 7.93-fold improvements). For seqall, the best effective energy efficiency is achieved by *lzop* -6 (2.21- and fold 6.56-fold improvements).



(a)

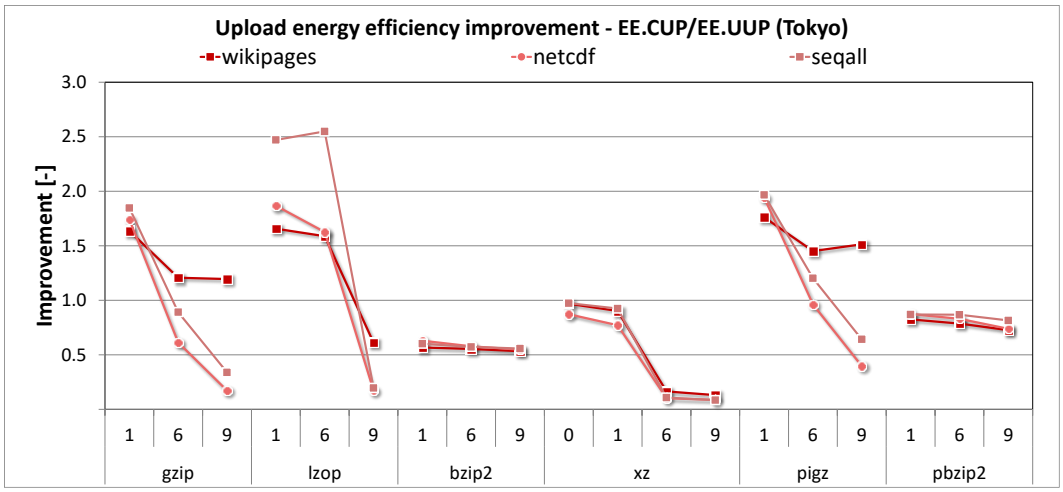


(b)

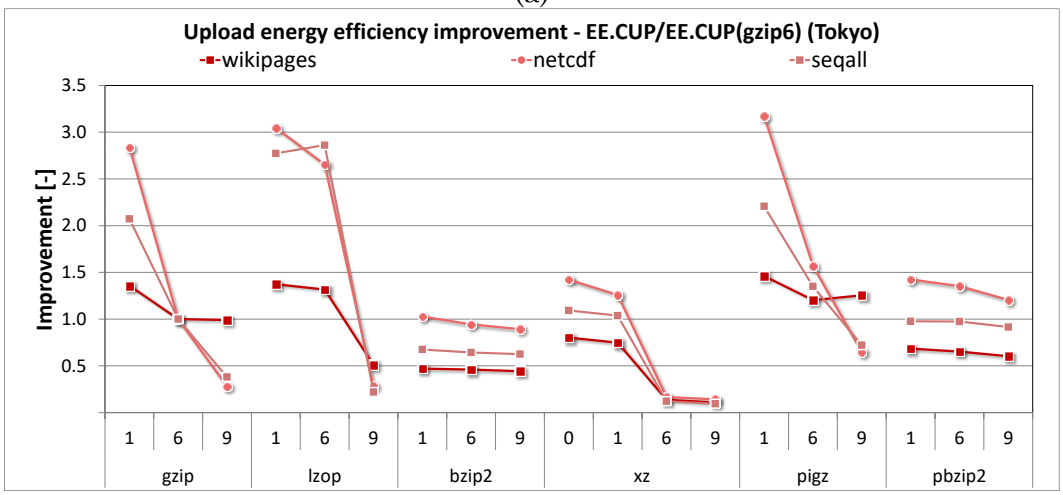
Figure 2.25 Upload energy efficiency improvement for upload of wikipages, netcdf, and seqall files to North Virginia on uncapped LAN connection

Figure 2.26 shows the effective upload energy efficiency improvements for all compressed upload modes relative to the corresponding uncompressed transfers (a) and the default compressed transfers with *gzip* -6 (b) to the Tokyo instance when using the uncapped LAN network. The uncompressed uploads achieve the effective energy efficiencies ranging from 1.02 MB/J to 1.11 MB/J for all three files. The default compressed upload with *gzip* -6 achieves the effective energy efficiencies of

1.33 MB/J for wikipages, 0.62 MB/J for netcdf, and 0.96 MB/J for seqall, all of which but one are below energy efficiency of uncompressed transfers. The utilities with the highest effective upload energy efficiencies are *pigz* and *lzop*. For wikipages, the best effective energy efficiency is achieved by *pigz* -1, offering 1.76- and 1.46-fold improvements in the effective energy efficiency over the uncompressed upload and the default compressed upload, respectively. Likewise, for netcdf, the best effective energy efficiency is achieved by *pigz* -1 (1.94- and 3.17-fold improvements). For seqall, the best effective energy efficiency is achieved by *lzop* -6 (2.55- and fold 2.86-fold improvements).



(a)



(b)

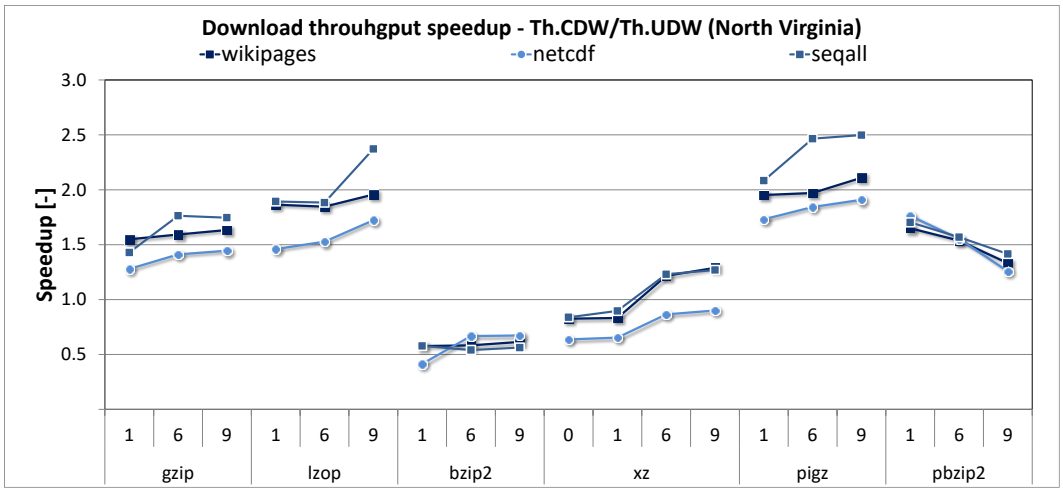
Figure 2.26 Upload energy efficiency improvement for upload of wikipages, netcdf, and seqall files to Tokyo on uncapped LAN connection

2.6.2.2 Download Examples

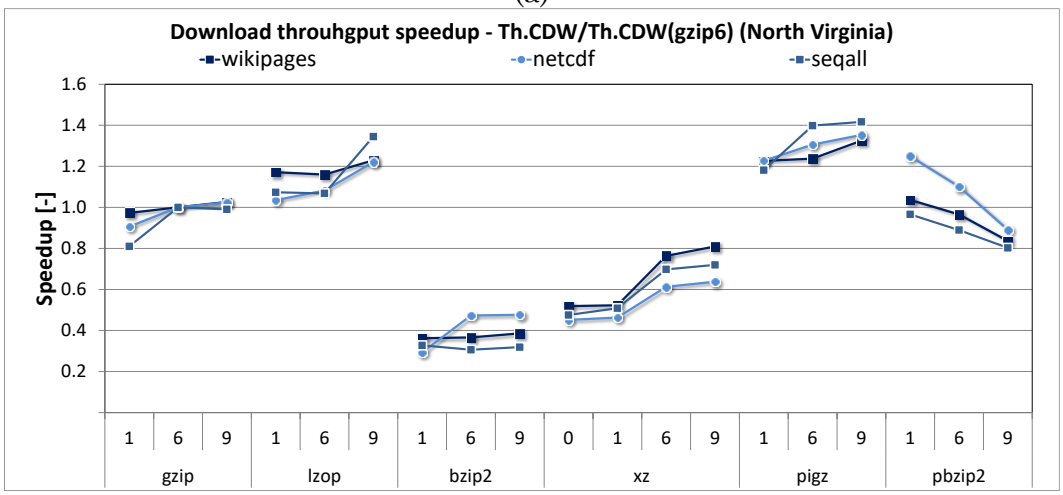
To evaluate the effectiveness of compressed downloads on the workstation, we consider uncompressed and compressed file downloads of the same files used for the upload example from the cloud instances in North Virginia and Tokyo. For compressed file downloads, all compressed versions of files are made available in the cloud. For each transfer mode, the total time and energy spent to get the uncom-

pressed file are measured to determine the effective download throughput and energy efficiency.

Download throughput. Figure 2.27 shows the effective download throughput speedups for all compressed download modes relative to the corresponding uncompressed transfers (a) and the default compressed transfers with *gzip* -6 (b) from the North Virginia instance when using the uncapped LAN network. The uncompressed downloads achieve the effective throughputs of 59.37 MB/s for wikipages, 66 MB/s for netcdf, and 70.34 MB/s for seqall. The default compressed downloads with *gzip* -6 achieve the effective throughputs of 94.54 MB/s for wikipages, 93.13 MB/s for netcdf, and 124.03 MB/s for seqall. The utility with the highest effective throughputs is *pigz*. For wikipages, the best effective throughput is achieved by *pigz* -9, offering 2.11- and 1.33-fold improvements in the effective throughput relative to the uncompressed download and the default compressed download, respectively. Likewise, for netcdf, the best effective throughput is achieved by *pigz* -9 (1.91- and 1.25-fold improvements). For seqall, the best effective throughput is achieved by *pigz* -9 (2.5- and 1.42-fold improvements).



(a)

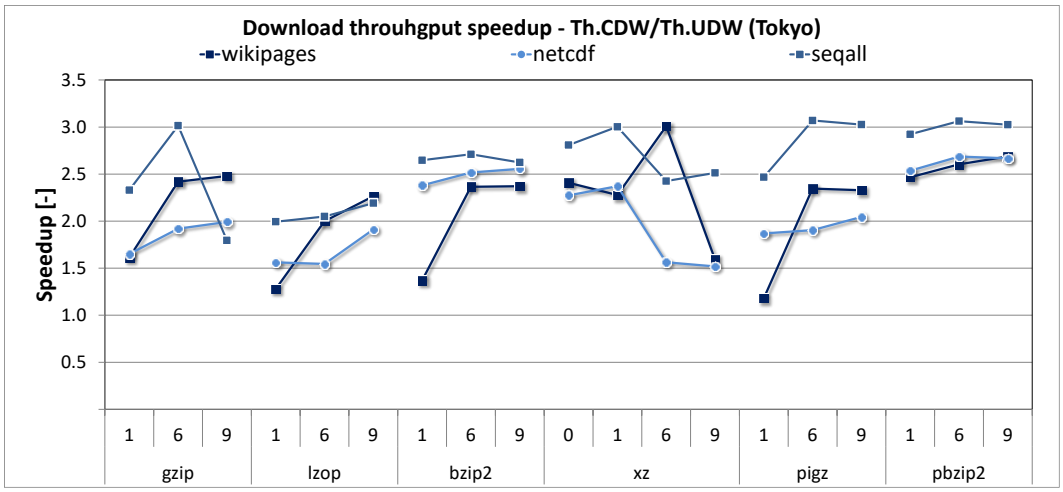


(b)

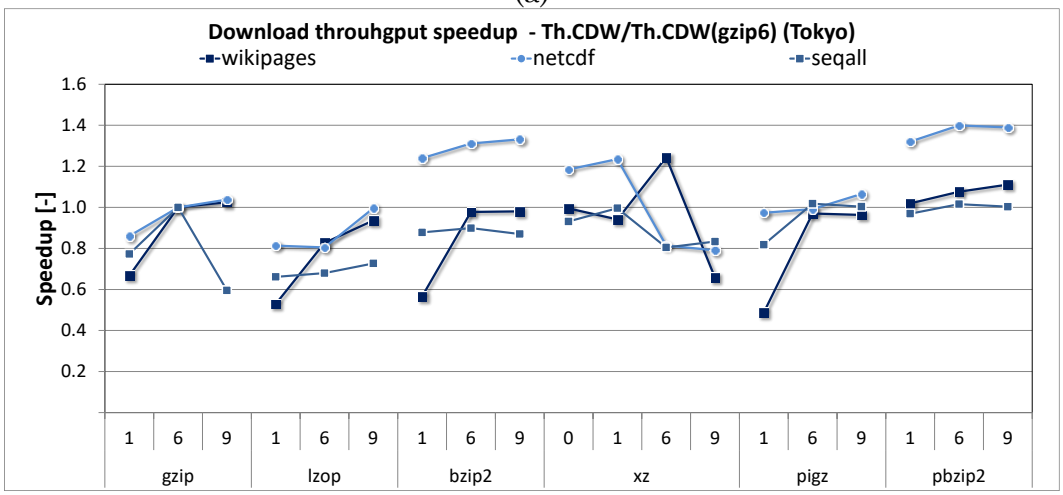
Figure 2.27 Download throughput speedup for download of wikipages, netcdf, and seqall files from North Virginia on uncapped LAN connection

Figure 2.28 shows the effective download throughput speedups for all compressed download modes relative to the corresponding uncompressed transfers (a) and the default compressed transfers with *gzip* -6 (b) from the Tokyo instance when using the uncapped LAN network. The uncompressed downloads achieve the effective throughputs of 3.01 MB/s for wikipages, 2.69 MB/s for netcdf, and 3.07 MB/s for seqall. The default compressed downloads with *gzip* -6 achieve the effective

throughputs of 19.76 MB/s for wikipages, 17.39 MB/s for netcdf, and 27.24 MB/s for seqall. The utilities with the highest download throughputs are *pigz*, *xz*, and *pbzip2* because they achieve fast decompression with parallelization, and in selected cases, high compression ratios. For wikipages, the best effective throughput is achieved by *xz* -6, offering 3.01- and 1.24-fold improvements in the effective throughput over the uncompressed download and the default compressed download, respectively. For netcdf, the best effective throughput is achieved by *pbzip2* -6 (2.69- and 1.4-fold improvements). For seqall, the best effective throughput is achieved by *pigz* -9 (3.07- and 1.02-fold improvements).



(a)

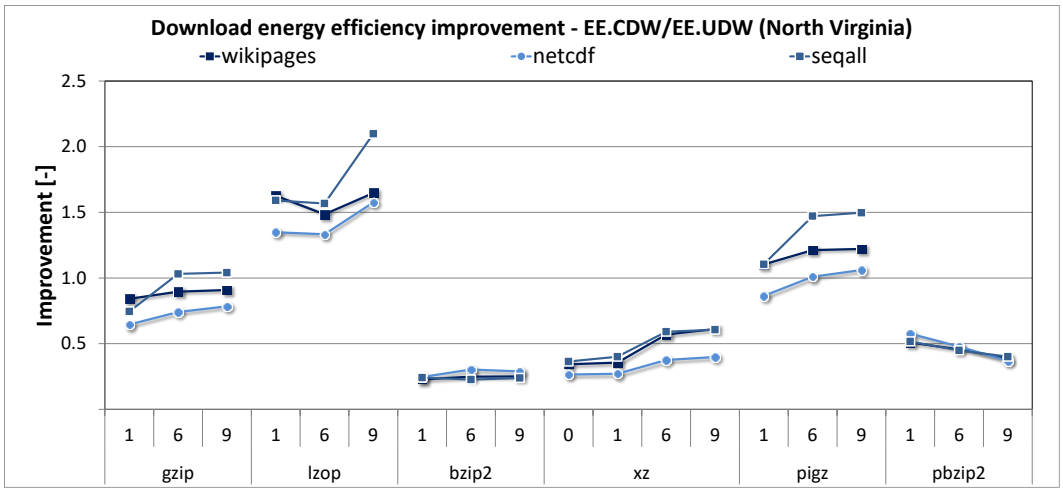


(b)

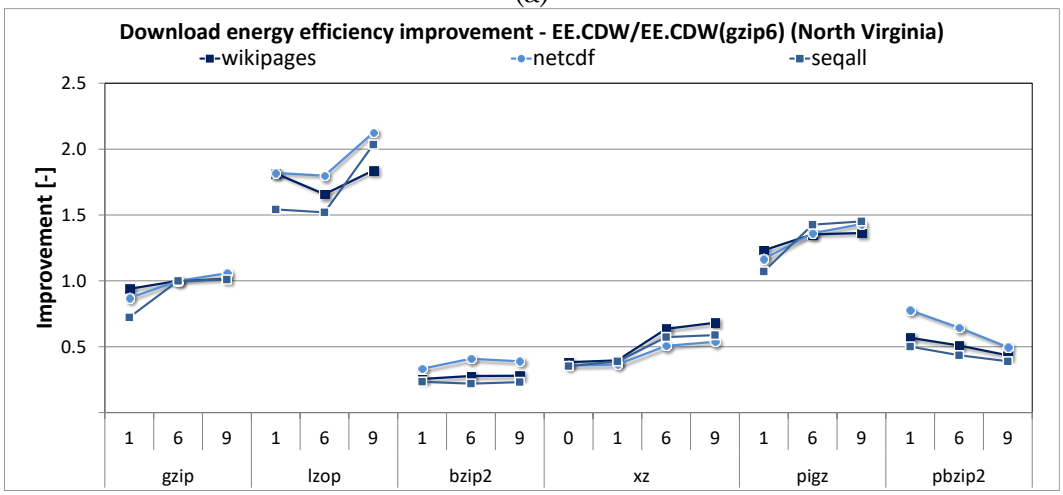
Figure 2.28 Download throughput speedup for download of wikipages, netcdf, and seqall files from Tokyo on uncapped LAN connection

Download energy efficiency. Figure 2.29 shows the effective download energy efficiency improvements for all compressed download modes relative to the corresponding uncompressed transfers (a) and the default compressed transfers with *gzip* -6 (b) from the North Virginia instance when using the uncapped LAN network. The uncompressed downloads achieve the effective energy efficiencies of 8.84 MB/J for wikipages, 9.63 MB/J for netcdf, and 9.73 MB/J for seqall. The default com-

pressed downloads with *gzip* -6 achieve the effective energy efficiencies of 7.92 MB/J for wikipages, 6.99 MB/J for netcdf, and 10.04 MB/J for seqall, all of which but one are below energy efficiency of uncompressed transfers. The utility with the highest effective download energy efficiencies is *lzop* -9. For wikipages, the best effective energy efficiency achieved by *lzop* -9 offers 1.65- and 1.84-fold improvements in the effective energy efficiency over the uncompressed download and the default compressed download, respectively. For netcdf, the best effective energy efficiencies are achieved by *lzop* -9 (1.58- and 2.13-fold improvements). For seqall, the best effective energy efficiencies are achieved by *lzop* -9 (2.1- and 2.03-fold improvements).



(a)

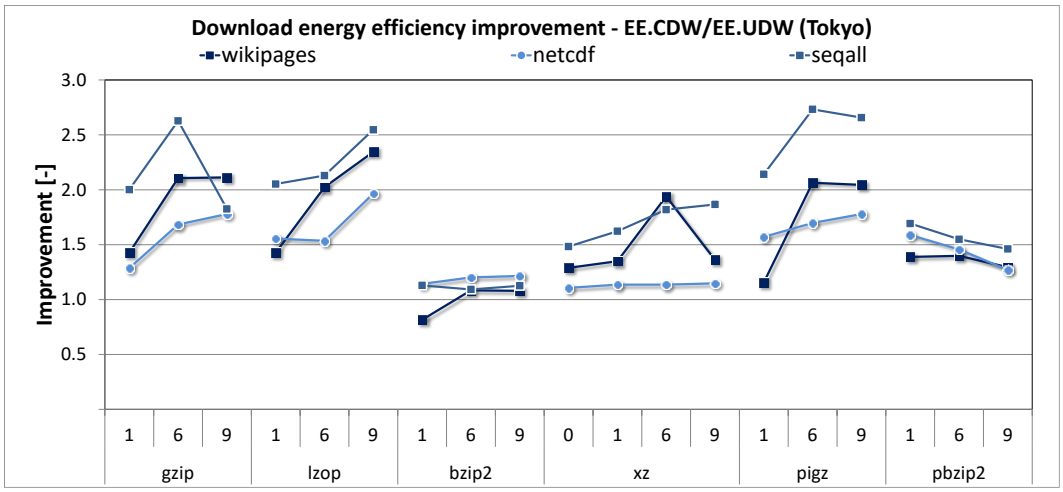


(b)

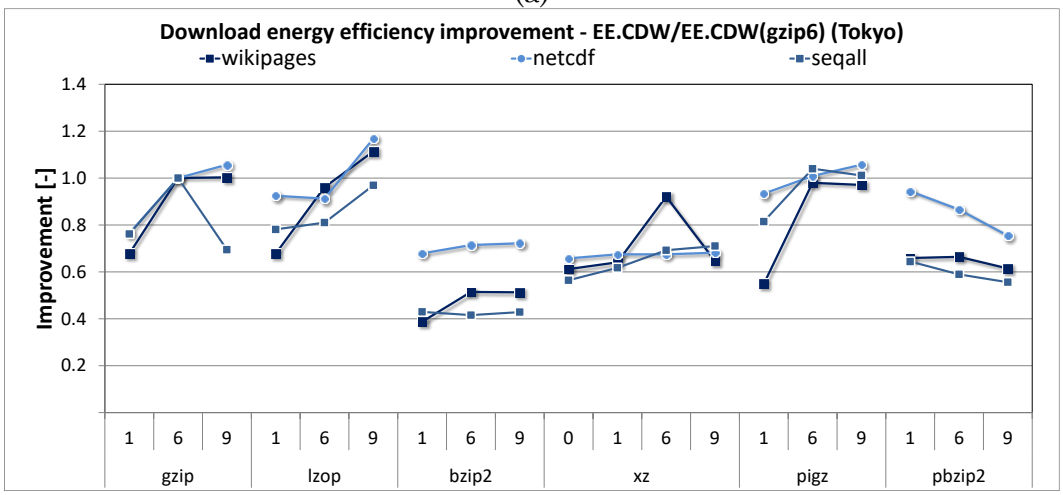
Figure 2.29 Download energy efficiency improvement for download of wikipages, netcdf, and seqall files from North Virginia on uncapped LAN connection

Figure 2.30 shows the effective download energy efficiency improvements for all compressed download modes relative to the corresponding uncompressed transfers (a) and the default compressed transfers with *gzip* -6 (b) from the Tokyo instance when using the uncapped LAN network. Because of lower network throughput and higher connection times, the uncompressed download achieves the effective energy efficiencies of 1.72 MB/J for wikipages, 1.85 MB/J for netcdf, and 1.82 MB/J

for seqall. The default compressed downloads with *gzip* -6 achieve the effective energy efficiencies of 3.61 MB/J for wikipages, 3.12 MB/J for netcdf, and 4.78 MB/J for seqall. The utilities with the highest effective download energy efficiencies are *lzop* -9 and *pigz* -6. For wikipages, the best effective energy efficiency is achieved by *lzop* -9, offering 2.35- and 1.11-fold improvements in energy efficiency over the uncompressed download and the default compressed download, respectively. For netcdf, the best effective energy efficiency is achieved by *lzop* -9 (1.97- and 1.17-fold improvements). For seqall, the best effective energy efficiency is achieved by *pigz* -6 (2.73- and fold 1.04-fold improvements)..



(a)



(b)

Figure 2.30 Download energy efficiency improvement for download of wikipages, netcdf, and seqall files from Tokyo on uncapped LAN connection

2.6.3 The Case for a Framework for Optimized Data Transfers

The two studies from above demonstrated that no single utility and compression level offers the best throughputs and energy efficiencies in all conditions. The file sizes, type of source data, edge device characteristics, and network conditions all impact the choice of best performing utilities. In addition, the case studies showed that the best performing combination of compression utility and compression level

provides a substantial increase in throughput and energy efficiency when compared to uncompressed data transfers or compressed data transfers that rely on the default *gzip* -6 compression. Whereas the case studies rely on exhaustive measurements in identifying the best performing combinations, the question is whether we can design a framework that will be able to select near best performing combinations in real-time, i.e., without incurring significant additional latency.

The proposed framework estimates effective throughputs and energy efficiencies for all available compression utilities and levels, before selecting an optimal or near optimal mode of data transfer. The estimation of effective throughput for each utility relies on the following:

- File information (e.g., file size, file type);
- Network parameters (maximum throughput, starting connection delay);
- Local (de)compression histories for files of given type;
- A set of models that describe uncompressed and compressed data transfers.

The estimation of effective energy efficiency relies on the same set of performance parameters, plus device characteristics and active peak currents for all (utility, level) combinations and for uncompressed transfers used in the additional set of equations to estimate energy efficiencies from the performance metrics (throughputs).

Once the estimations are computed, the (utility, level) pair with the highest estimated throughput or energy efficiency will be compared to the throughput or energy efficiency of uncompressed data transfer. If the throughput or energy efficiency of the compressed data transfer exceeds the throughput or energy efficiency of un-

compressed transfers, the data will be transferred to a remote location with the selected utility and level. Otherwise, the uncompressed data transfer is performed.

CHAPTER 3

RELATED WORK

Growing dependency of users on services delivered through cloud makes optimization of data transfers a top priority for mobile devices, workstations, as well as in the cloud services dealing with data deluge. For mobile devices, optimization of data transfers is driven by several key factors, including (i) user convenience favoring of low latency of data, (ii) low network throughputs with 3G/4G connection, (iii) limited energy capacity of batteries, and (iv) cost and data cap of mobile data plans. For workstations, servers, and cloud infrastructure it is driven by sever factors, including (i) network throughput limitations due to high network traffic and routing or infrastructure limitations, (ii) increasing costs of cloud resources and transfer fees, and (iii) increasing power and cooling costs in data centers and server farms.

A number of different studies explore new ways for increasing performance and overall energy efficiency for both mobile and workstation systems. The proposed and existing solutions for managing power consumption include (i) heterogeneous ARM big.LITTLE architecture and Heterogeneous Multi-Processing (HMP) scheduler on state-of-the-art mobile devices and SoCs (e.g., Samsung Exynos Octa 8860, Qualcomm Snapdragon 820, and NVIDIA Tegra X1), (ii) optimizations using frequency and voltages scaling, (iii) cloud offloading schemes [66], [67], and (iv) use of lossless data compression for data communication. The proposed solutions for finding new ways for power optimizations on mobile and workstation systems include

custom measurement environments for capturing power traces and logging to capture execution history [68]–[72] and run-time power modeling and energy estimation by utilizing performance counters, call tracing, and system observers [73]–[76].

The rest of this chapter covers the related work in the area of lossless data (de)compression utilities used in optimization of data communication on mobile and workstation systems (Section 3.1) and the related work in the area of power proofing and energy estimation on mobile and workstation systems (Section 3.2).

3.1 Lossless Data Compression on Mobile and Cloud Systems

We are aware of several related studies that use data compression as a solution for some of the emerging problems in the mobile and cloud computing, including data management and data transfer. Nicolae et al. [77] and Harnik et al. [78] present methods for deciding whether to perform compression or not with a goal to increase effectiveness of data transfers from the local nodes to the cloud. The presented solutions only explore a selection of compression algorithms or utilities (e.g., *gzip* and *bzip2*) based on the estimation of file entropy. The presented estimations rely either on pre-compression of multiple samples across the entire file or on pre-compression of the beginning of the file. Harnik et al. [78] introduce an approach to sampling of large files, which provides a low error in estimation of compression ratio, as well as different methods for smaller files, which is more prone to estimation errors. Raskhodnikova et al. focus on mathematical estimation of compressibility of text files for specific compression algorithms by analyzing the content of the entire file [79].

General compression utilities and their effectiveness in data communication have been studied in mobile systems. A study conducted a decade ago investigates

data compression in the context of energy efficiency on embedded and mobile systems [23], [24]. Our preliminary studies have also focused on the comparison of performance and energy efficiency of general purpose compression utilities in mobile and mHealth applications [17]–[20]. Other papers focus on performance comparison of general purpose, as well as domain-specific compression utilities, e.g., for genomics data files [80], [81].

The work presented in this dissertation differs from prior works as follows. First, a larger set of compression utilities is considered and evaluated. We have also shown that compression ratio alone is not the deciding factor in the effective throughput or energy efficiency of compressed uploads or downloads. Instead, our proposed framework relies on a range of parameters to estimate effectiveness of transfer modes, including performance of each particular compression (utility, level) pair, input file information, network parameters, as well as device characteristics. We derive analytical estimation models for prediction of the effective throughput [21], energy efficiency [22], as well as modes for estimating energy efficiency from the performance metrics for compressed uploads and downloads. In this work we avoid the computational overhead due to performing sampling methods on part of the file to estimate compression ratio or throughput. Such pre-compression, as presented by Nicolae et al. and Harnik et al., can be inaccurate and time and energy consuming. Instead, we use predictability of file entropy by relying on previously extracted prediction data for local (de)compression of all available compression utilities on the system for all considered file types.

3.2 Power Profiling and Energy Estimation

This section discusses previously conducted studies in the field of power measurements, profiling, and energy estimation on mobile (Section 3.2.1) and workstation (Section 3.2.2) systems.

3.2.1 Mobile Devices

Runtime power measurements on real mobile devices running common software platforms such as Android, iOS, Tizen, or Windows Phone are important for both researchers and mobile application developers. Measurement frameworks can capture complex interactions between hardware and software stacks that become more and more sophisticated with the introduction of systems-on-a-chip (SoCs) with multiple processor cores and a number of customized hardware accelerators. Measurements on real devices can help research studies that target power optimizations or those that target developing analytical models for energy estimation based on parameters derived from real platforms. For mobile developers, adding a power perspective to application debugging and testing may guide optimizations that will result in more energy-efficient mobile applications.

We are aware of several related studies that investigate energy efficiency on mobile devices using custom measurement environments for capturing power traces and logging to capture execution history [68]–[71], [82]. A study by Shye et al. [68] relies on power models and extended activity logging to generate power schemes which can provide substantial energy saving across the entire system while maintaining user satisfaction. Their study was based on Android G1 running Android 1.0 firmware. They also used a setup based on a shunt resistor to capture power traces and a custom logger to generate activity traces. However, their setup offered a lim-

ited sampling frequency of only 1 Hz. Rice and Hay [69], [70] evaluated energy efficiency of Android-based G1, Magic, and Hero handsets using their custom measurement setup. Their setup includes a replacement battery and a high-precision shunt resistor placed in series on the power line and an NI data acquisition device that samples voltage drop across the resistor. Their excellent studies focused on measurement-based evaluation and optimization of wireless communication in mobile handsets. A similar setup is used in our prior study focusing on energy-efficiency of Pandaboard and Raspberry Pi development platforms that run Linux operating system [71]. The setup included features to allow automated power measurements for a number of profiled applications. Carroll and Heiser quantified energy consumption of each component in a mobile device by performing rigorous tests and then simulating a number of usage scenarios on mobile devices [82].

However, since hardware modifications to support direct energy measurements are not always possible or desirable, several other papers performed studies focused on power profiling and power estimation by utilization of performance counters, call tracing, system observers and subsequent generation of system-specific models for estimation of power consumption. Bircher and John used processor performance counters and system-specific models to estimate consumption of CPU, memory, disk, and I/O [73]. Pathak et al. [74], [75] and Li and John [76] used system call tracing and known observations of the system to generate models that can perform run-time power estimation with fine-grained measurements.

Whereas prior studies focused on capturing power traces on smartphones [68]–[71], [82], they relied on manual control and post-processing for synchronization of power traces with events in profiled programs or focused on early smartphones and software platforms. In addition, they relied on hardware setups

that required inserting a shunt resistor on the power supply line, thus introducing a slight deviation in the power supply of the device under test. The setup for automated power profiling [56], [57] used in preliminary studies and framework evaluation on mobile devices for this dissertation offers several advantages over the setups introduced in [68]–[71], [82]. For example, we utilize Android Debug Bridge (adb) to remotely control the mobile device and launch script command files for unobtrusive power measurements. Next, we use network time synchronization protocol to precisely capture activities on the mobile device and synchronize the current samples collected on the workstation with these activities. Our use of the battery simulator eliminates any voltage changes across the shunt resistor due to drainage of the battery. Additionally, *mLViewPowerProfile* offers flexible control and automation of experiments.

For this dissertation, energy estimation is also performed through the use of models which estimate energy efficiency of uncompressed and compressed transfers by utilizing prediction tables filled with performance data and several device-specific characteristics. Whereas prior studies focused on the use of performance counters and call tracing to generate power estimation models [73]–[76], the current state of our models and energy estimations used in the framework does not rely on performance counters. However, it is one of the possible future improvements that can provide better energy estimation of uncompressed and compressed data transfers and improvement to framework adaptation on new devices and platforms.

3.2.2 Workstations

The statement that hardware modifications to support direct energy measurements are not always possible or desirable holds true for workstation and server

computer systems, with some cases where hardware modifications can be almost impossible to perform. This is why several prior studies focused power estimation and power characterization through use of performance counters and system call tracing on Intel and AMD architecture workstations, as in study by Bircher and John [73]. Since modern Intel microprocessors on workstations, starting with Sandy Bridge architecture, include the RAPL interface [83], [84], energy estimates have been validated by Intel to verify that they closely follow the actual energy used [51].

CHAPTER 4

FRAMEWORK OVERVIEW AND DESIGN

This chapter gives an overview of the framework for intelligent file transfers between edge devices and the cloud. Section 4.1 gives an overview of the proposed framework and its main components and Section 4.2 discusses the design of the framework components.

4.1 Framework Overview

This section gives an overview of the proposed framework for intelligent file transfers between edge devices (mobile, workstation) and the cloud. The next two sub-sections give an overview of the framework operation for three transfer methods from the perspective of an edge device: optimized file uploads (Section 4.1.1) and optimized file downloads with and without on-demand compression (Section 4.1.2).

4.1.1 Optimized File Uploads

Figure 4.1 illustrates a system view of optimized file uploads initiated on an edge device. A software agent running on the device is responsible for selecting an effective transfer mode for each file upload. The agent uses two modes of optimization, one uses the effective throughput and another uses energy efficiency as a measure of effectiveness. The mode of optimization can be specified by the user (human or machine) during each file upload. Additionally, the agent can be configured to use a combined metric for additional modes of optimization, such as a product of

the effective throughput and the energy efficiency. To select the most effective transfer mode, the agent relies on the following inputs: (i) file information (file type and file size); (ii) network connection parameters; (iii) analytical models describing the effective throughput for uncompressed file uploads as well as for compressed file uploads for all (utility, level) combinations available on the device; (iv) device characteristics and active peak currents for all (utility, level) combinations and uncompressed uploads used in estimating effective energy efficiency for energy-efficient mode of optimization; and (v) history-based prediction tables that for a given uncompressed file predict the compression ratio and the local compression throughput for all compression utilities. The energy-efficiency is estimated from the throughput and active peak currents for all (utility, level) combinations and uncompressed transfers.

On an upload request, the agent performs a query on the prediction table with the file size, type, and network connection parameters as inputs. The query produces estimated compression ratios and local compression throughputs for each (utility, level) pair supported on the device. These estimates are then used in the analytical models to calculate estimates for either the effective compressed upload throughputs or upload energy efficiencies for each (utility, level) pair, depending on the operating mode. For throughput mode, the effective throughput of the best performing combination of (utility, level) pair is compared to the effective throughput of the uncompressed upload. If it offers a higher throughput, the compressed upload with the (utility, level) pair is selected. Similarly, for energy-efficient mode, the estimated energy efficiency of the best performing combination of (utility, level) pair is compared to the estimated energy efficiency of the uncompressed upload. If it offers higher energy efficiencies, the (utility, level) pair is selected. The agent then simul-

taneously initiates the selected compressed upload. Otherwise, the file is transferred uncompressed.

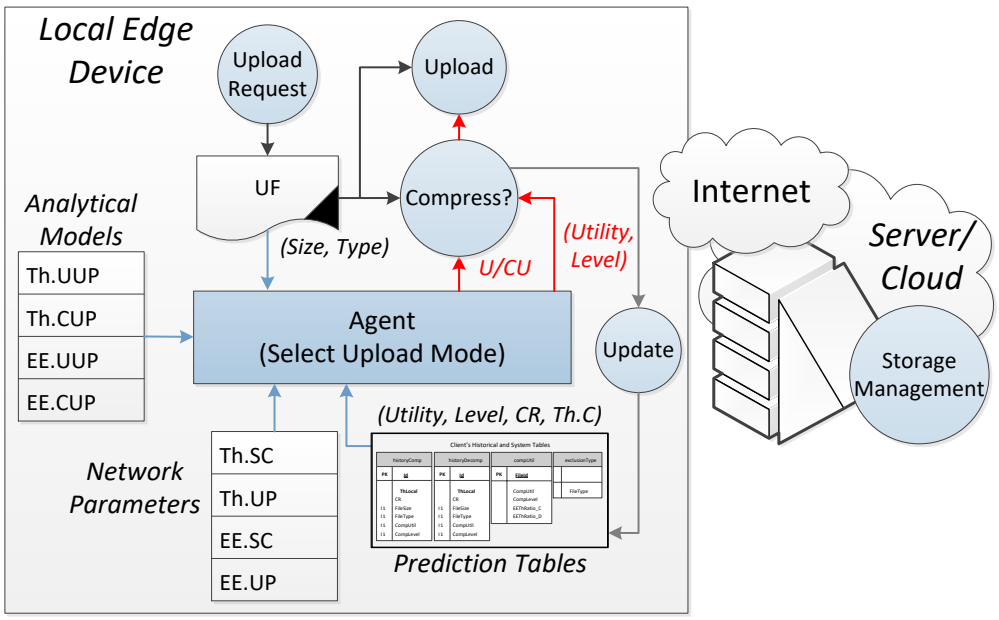


Figure 4.1 System view of optimized data file uploads

The expected savings due to compressed file uploads should exceed by far the time or energy the agent spends in the selection process. The agent maintains the parameters that characterize the network connection: the network throughput, *Th.UP*, and the time to set up a network connection, *T.SC*. To avoid imposing an additional overhead to the current file upload, we assume that these parameters are acquired prior to upload – e.g., an agent can periodically probe the network connection of interest. Once the transfer has been completed, the agent determines the compression ratio and the local compression throughput and creates a new entry in the prediction tables to inform future queries. Additionally, after a certain period of

time or the number of file transfers, the agent reduces the number of entries in the prediction tables by averaging similar entries to simplify and speed up future queries.

4.1.2 Optimized File Downloads

Figure 4.2 and Figure 4.3 illustrate a system view of optimized file downloads initiated from an edge device. A software agent running on the device initiates a download request by sending the device *id*, file name, and current network parameters to the server. The software agent running on the server is then responsible for selecting an effective transfer mode for the edge device for each file download. The server agent either sends already compressed file (Figure 4.2) or performs an on-demand compression of uncompressed file (Figure 4.3). In both cases, the agents use one of the two modes of optimization, for throughput or for energy efficiency. As in uploads, the mode of optimization can be specified by the user (human or machine) for each file download. Similarly, the agents can be configured to use combined metric for additional modes of optimization, such as a product of the effective throughput and the energy-efficiency.

To select the most effective transfer mode, the agent on the server relies on the following inputs: (i) device *id* sent by the edge device to make device specific selection; (ii) network connection parameters sent by the edge device; (iii) file information (file type and file size); (iv) analytical models describing the effective throughput for uncompressed file downloads as well as for compressed file downloads (with or without on-demand compression) for all (utility, level) combinations; (v) device characteristics and active peak currents for all (utility, level) combinations and uncompressed downloads; and (vi) history-based prediction tables that predict

the compression ratio and the local decompression throughput for a given compressed file.

Once the edge device sends the request to the server, the server agent performs a query on the prediction table with the edge device *id*, the uncompressed file information (size and type), and edge device network connection parameters as inputs. If the server maintains compressed versions of the requested file, the actual compression ratios are known and do not have to be predicted (to lower the overhead, the server agent stores compression ratios for all managed files). Thus, the query retrieves actual compression ratios and estimated local decompression throughputs for each (utility, level) pair supported on the requesting device. In case of on-demand compression, compression ratios have to be estimated from the prediction table. Thus, the query produces estimated compression ratios, the local decompression throughputs for each (utility, level) pair supported on the requesting device, and the local compression throughputs for each (utility, level) pair supported on the server in the cloud. These estimates are then used in the analytical models to calculate either the estimated effective compressed download throughputs or energy efficiencies for each (utility, level) pair. The server agent then compares the estimated metrics for the best performing compressed downloads to the uncompressed downloads and selects an appropriate transfer mode that minimizes the metric of interest.

The expected savings due to compressed file downloads should exceed by far the time the server agent spends in the selection and possible compression tasks. The mobile device agent maintains the parameters that characterize the network connection: the network throughput, *Th.DW*, and the time to set up a network connection, *T.SC*. To avoid imposing an additional overhead to the current file down-

load, we assume that these parameters are acquired prior to download – e.g., an agent can periodically probe the network. Furthermore, instead of sending network parameters on each download request, a mobile device can instead establish a cookie for the server to access for an extended period of time (e.g., web session, application session). Once the transfer has been completed, the edge device agent determines the compression ratio and the local decompression throughput and creates a new entry in the prediction tables to inform future downloads. Additionally, after a certain period of time or the number of file transfers, the edge device agent reduces the number of entries in the prediction tables by averaging similar entries to simplify and speed up future queries.

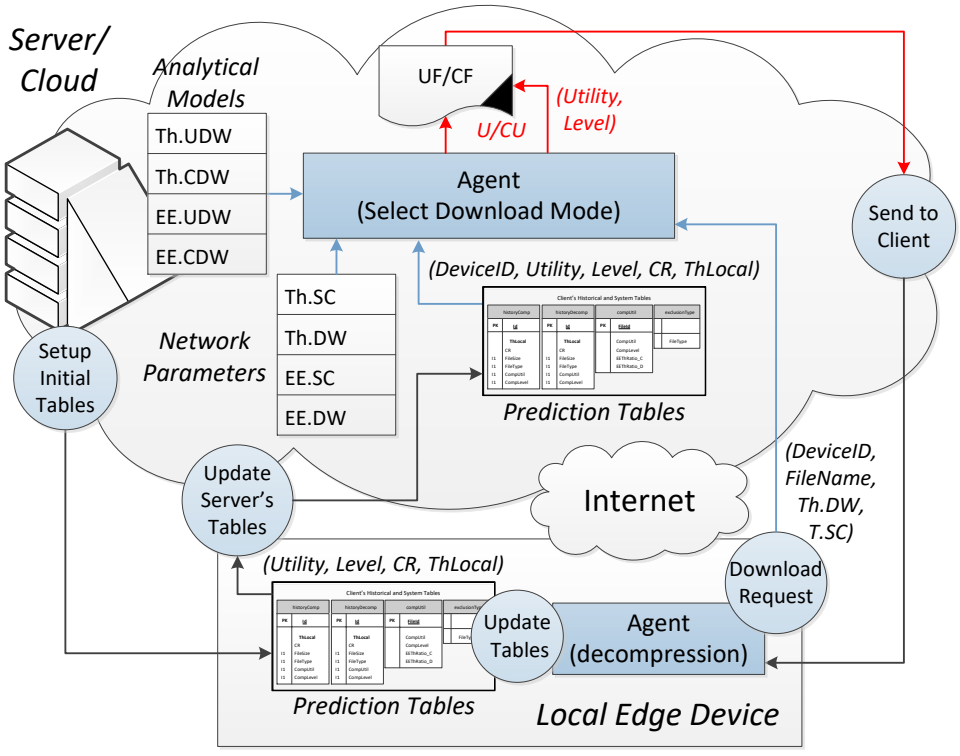


Figure 4.2 System view of optimized data file downloads

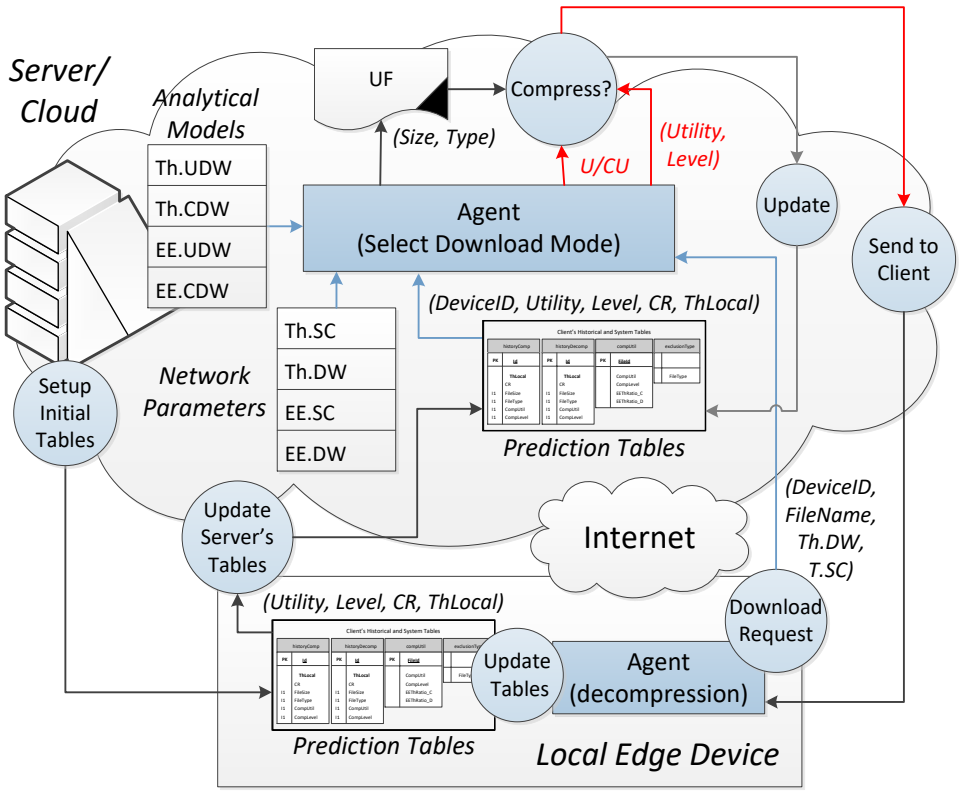


Figure 4.3 System view of optimized data file downloads with on-demand compression

4.2 Framework Components

The first step in creating a framework for optimal data transfers is the development of analytical models by which metrics characterizing compressed and uncompressed data transfers can be estimated. Analytical models should be able to accurately estimate the effective throughput and energy efficiency for different modes of data transfers.

The rest of this section discusses the design of the framework for optimal data transfers. Section 4.2.1 introduces modeling of uncompressed file transfers, including analytical models for performance, energy metrics and estimated energy metrics from performance (Section 4.2.1.1), experimental verification of proposed models over a number of communication channels (Section 4.2.1.2), and method for deriving network parameters (Section 4.2.1.3). Section 4.2.2 introduces modeling of compressed file transfers, including performance limits (Section 4.2.2.1), energy limits (Section 4.2.2.2), a piping model for compressed file transfers (Section 4.2.2.3), and a method for estimation of energy metrics from performance metrics and device characteristics (Section 4.2.2.4). Section 4.2.3 discusses the acquisition of historical prediction data to describe the performance of local (de)compression tasks. Section 4.2.4 presents the basic data table structure (Section 4.2.4.1), which includes prediction and other tables. Section 4.2.4.2 describes the life cycles of data tables during different stages of the framework operation. Finally, a number of database optimizations are presented in Section 4.2.4.3.

4.2.1 Modeling Uncompressed File Transfers

Section 4.2.1.1 describes the analytical models for effective throughput and energy efficiency of uncompressed file transfers. Section 4.2.1.2 presents experi-

mental verification of the proposed analytical models on a mobile device and workstation over the selection of communication channels (LAN, WLAN, and 3G/4G), network throughputs, and transfers to both local and cloud servers. Section 4.2.1.3 describes the derivation of network parameters to characterize the communication channel using the proposed framework and a limited number of measurements.

4.2.1.1 Analytical Models for Uncompressed File Transfers

Execution time and throughput. The total time to perform a file transfer includes edge device overhead time, network connection setup time, file transmission time, and cloud overhead time. In case of uncompressed file uploads, the edge device and cloud overheads can be ignored. Thus, the total time of an uncompressed file upload, *T.UUP*, includes the time to set up a network connection, *T.SC*, and the file transmission time, *T.UP*, as shown in Equation (4.1). If we know the network upload throughput, *Th.UP*, the file transmission time for upload can be calculated as the ratio between the file size and the network upload throughput, $T.UP=US/Th.UP$. Similarly, the total time of an uncompressed file download, *T.UDW*, includes *T.SC* and the file transmission time, *T.DW*, as shown in Equation (4.2). The file transmission time for download can be calculated as $Th.DW=US/Th.DW$, where *Th.DW* is the network download throughput.

The effective upload throughput is calculated as the uncompressed file size in megabytes, *US*, divided by the total time to upload the file, $Th.UUP=US/T.UUP$. The effective download throughput, *Th.UDW*, is calculated as the uncompressed file size, *US*, divided by the total time to download the file, $Th.UDW=US/T.UDW$. Equations (4.3) and (4.4) show the expressions for the effective upload and download throughputs, respectively. The effective throughputs depend on the file size, the

time to set up the network connection, and the network upload and download throughputs. The effective throughputs, *Th.UUP [Th.UDW]*, reach the network throughputs, *Th.UP [Th.DW]*, when transferring large files. In case of smaller files, the time to set up the network connection limits the effective throughput.

$$T.UUP = T.SC + T.UP = T.SC + US/Th.UP \tag{4.1}$$

$$T.UDW = T.SC + T.DW = T.SC + US/Th.DW \tag{4.2}$$

$$Th.UUP = \frac{Th.UP}{1 + Th.UP \cdot T.SC/US} \tag{4.3}$$

$$Th.UDW = \frac{Th.DW}{1 + Th.DW \cdot T.SC/US} \tag{4.4}$$

Total energy and energy efficiency. The total energy at the edge device needed to perform a file transfer includes edge device energy overhead, the energy to set up a network connection, and the energy to perform the file transmission. In a case of uncompressed file uploads, the edge device overhead can be ignored. Thus, the total energy cost of an uncompressed data file upload, *ET.UUP*, includes the energy to set up a network connection, *ET.SC*, and the energy to perform file transmission, *ET.UP*, as shown in Equation (4.5). If we know the network upload energy efficiency, *EE.UP*, the file transmission energy can be calculated by dividing the file size with the network upload energy efficiency, $ET.UP=US/EE.UP$. Similarly, the total energy of an uncompressed data file download, *ET.UDW*, includes *ET.SC* and the energy of file transmission, *ET.DW*, as shown in Equation (4.6). The file transmission energy can be calculated as $ET.DW=US/EE.DW$, where *EE.DW* is the network download energy efficiency.

The effective upload energy efficiency is calculated as the uncompressed file size divided by the total energy to upload the file, *EE.UUP=US/ET.UUP*. The effective download energy efficiency is calculated as the uncompressed file size divided by the energy to download the file, *EE.UDW=US/ET.UDW*. Equations (4.7) and (4.8) show the expressions for the effective upload and download energy efficiencies, respectively, as the functions of the file size, the energy to set up the network connection, and the network upload and download energy efficiencies. The effective energy efficiencies, *EE.UUP [EE.UDW]*, reach the network energy efficiencies, *EE.UP [EE.DW]*, when transferring very large files. In case of smaller files, the energy to set up the network connection limits the effective energy efficiency.

$$ET.UUP = ET.SC + ET.UP = ET.SC + US/EE.UP \tag{4.5}$$

$$ET.UDW = ET.SC + ET.DW = ET.SC + US/EE.DW \tag{4.6}$$

$$EE.UUP = \frac{EE.UP}{1 + EE.UP \cdot ET.SC/US} \tag{4.7}$$

$$EE.UDW = \frac{EE.DW}{1 + EE.DW \cdot ET.SC/US} \tag{4.8}$$

Estimated energy efficiency. Since the energy consumed and energy efficiency cannot be measured as easily as the execution time and throughput, an alternative way is to estimate the energy consumed from the existing throughput models. To do that, device characteristics such as device battery voltage, V_{BS}, device idle current, I_{idle}, and maximum active peak currents, $I.UP_{delta}$ *[*$I.DW_{delta}$*]*, for data file upload and download have to be known. Thus we would like to estimate energy-based network parameters, *EE.UP [EE.DW]* and *ET.SC*, from the existing performance parameters, *Th.UP [Th.DW]* and *T.SC*. Those parameters can be extracted ahead of

time for each device, or per class of devices that share similar hardware characteristics (e.g., low-end and high-end mobile devices, low-end and high end workstations).

The total energy to perform a file transfer is the sum of idle energy calculated as the product of device battery voltage, V_{BS}, the idle current, I_{idle}, and the total time to perform the file transfer, *T.UUP [T.UDW]*), and the energy overhead, calculated as the product of the device battery voltage, the maximum active peak current, $I.UP_{delta}[I.DW_{delta}]$, and the total time to perform the file transfer. The final expressions are shown in Equation (4.9) and (4.10). The estimated network energy efficiency for upload, *EE.UP*, is calculated as the uncompressed file size, *US*, divided by total energy consumed in Equation (4.9). The final expression for estimated network energy efficiency for upload is shown in Equation (4.11), as a function of the effective upload throughput, the device battery voltage, and of idle and maximum active peak currents. The estimated network energy efficiency for download is calculated similarly and shown in Equation (4.12). The estimated energy to set up a network connection, *ET.SC*, is calculated as shown in Equation (4.13). This expression is due to the observation that the current gradually rises up to the active current during the network setup, which can be calculated as the area of the right-angled triangle.

$$ET.UUP = V_{bs} \cdot T.UUP \cdot I_{idle} + V_{bs} \cdot T.UUP \cdot I.UP_{delta} \tag{4.9}$$

$$ET.UDW = V_{bs} \cdot T.UDW \cdot I_{idle} + V_{bs} \cdot T.UDW \cdot I.DW_{delta} \tag{4.10}$$

$$EE.UP = \frac{US}{V_{BS} \cdot T.UUP \cdot (I_{idle} + I.UP_{delta})} = \frac{Th.UP}{V_{BS} \cdot (I_{idle} + I.UP_{delta})} \tag{4.11}$$

$$EE.DW = \frac{US}{V_{BS} \cdot T.UDW \cdot (I_{idle} + I.DW_{delta})} = \frac{Th.DW}{V_{BS} \cdot (I_{idle} + I.DW_{delta})} \tag{4.12}$$

$$ET.SC = V_{BS} \cdot T.SC \cdot (I_{idle} + \frac{1}{2} \cdot I.UP_{delta}[I.DW_{delta}]) \tag{4.13}$$

4.2.1.2 Experimental Verification of Analytical Models

To verify the models described by the equations from above, we perform a set of measurement-based experiments on a mobile device and a workstation as follows. For the mobile device, the OnePlus One smartphone is used to initiate a series of file uploads to a local server and a series of file downloads from a local server. For each file transfer, the execution time and the energy consumed are measured using a measurement setup for edge devices that involves a battery simulator described in Section 2.4.2 [19], [56], [57]. During first two experiments, the smartphone is connected to the Internet over its WLAN interface, and file transfers are conducted using an encrypted network channel (*ssh*) and non-encrypted network channel *(wget/FTP)*. The file sizes are set to vary from 1 KB to 100 MB. Note: the files are created using the Linux *dd* command as shown in Figure 4.4. The command utilizes the Linux input device */dev/zero*, a special 'file' that provides nulls (0x00) for all data read from it. The upload and download experiments are repeated for four network throughputs set to $Th.UP = Th.DW = 0.5$ MB/s, 2 MB/s, 3.5 MB/s, and 5 MB/s. During the third experiment, the smartphone is connected to the Internet over its 3G/4G network interface, and file transfers are repeated using an encrypted network channel (*ssh*) with uncapped network throughput.

```
dd if=/dev/zero of=output.dat  bs=1M  count=24
```

Figure 4.4 Creating 1M file using input device */dev/zero*

For the workstation, the Dell PowerEdge T110 II is used to initiate a series of file uploads to a local server and to the cloud instances in North Virginia and Tokyo

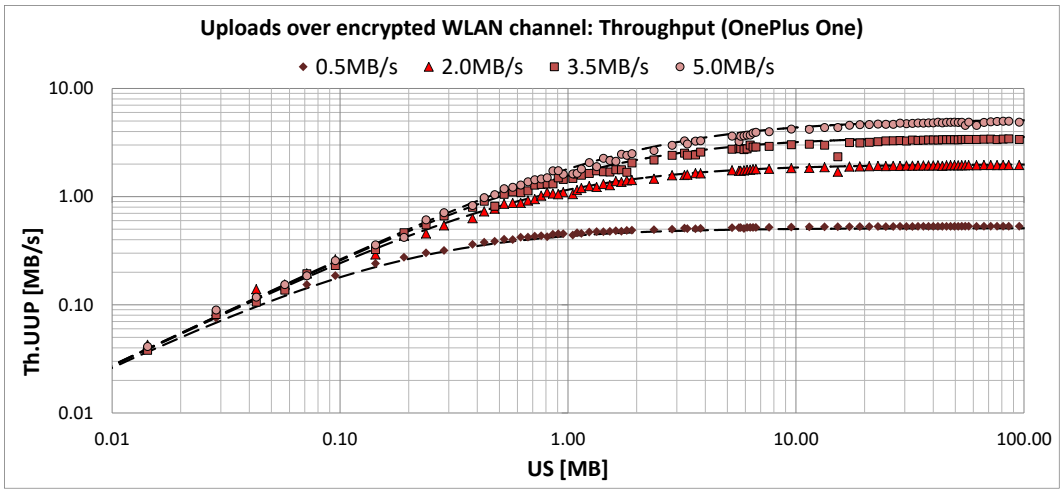
and a series of file downloads from a local server and from the cloud instances in North Virginia and Tokyo. For each file transfer, the execution time is measured using Linux's time and date utility with nanosecond precision. For transfers to and from the local server, the same set of generated files, used by OnePlus One (1 KB to 100 MB), is used to perform the first set of experiments with a local server. The upload and download experiments are repeated for four network throughputs set to $Th.UP = Th.DW$ = 0.5 MB/s, 2 MB/s, 3.5 MB/s, and 5 MB/s. For the second set of experiments with transfers to and from the cloud, subsets of files are selected from three datasets introduced in Table 2.4 (20 files from each dataset). The upload and download experiments are repeated using three network throughputs set to $Th.UP$=$Th.DW$= uncapped, 5 MB/s and 2 MB/s.

The rest of this subsection will present results of conducted experiments on the OnePlus One, with WLAN and 3G/4G network connection, and results of conducted experiments on the workstation, while transferring files to and from the local server with LAN network connection, and while transferring files to and from the cloud instances with LAN network connection. The experiments will demonstrate how network parameters depend on the type of connection (encrypted or non-encrypted), available network throughput, and finally, the location of a server or a cloud instance relative to the edge device requesting file transfers.

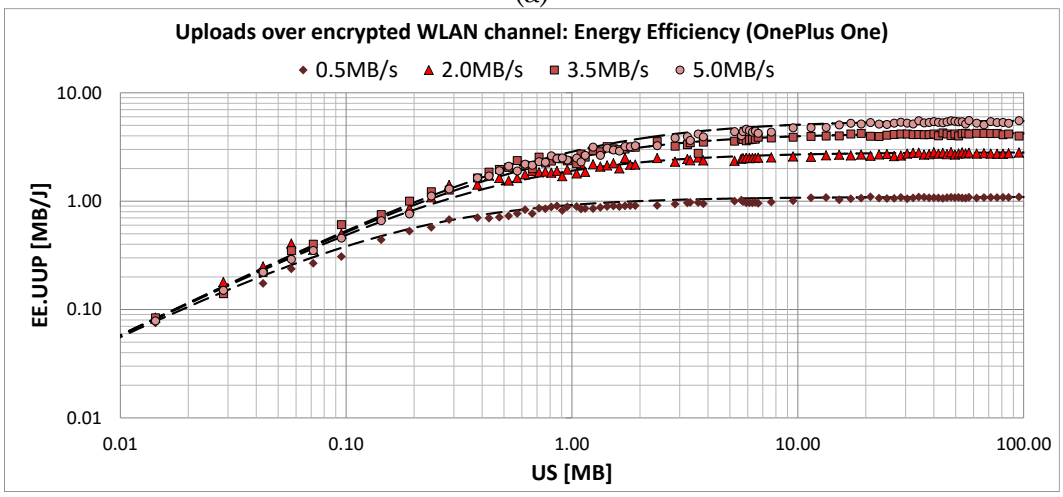
Mobile device (WLAN). Figure 4.5 (a) shows the measured effective throughputs for uncompressed uploads over the encrypted WLAN channel as a function of the file size, US, and the network connection throughput for uploads, $Th.UP$. The plots show that the effective throughput saturates for the larger files, reaching the network connection throughput, i.e., $Th.UUP$=$Th.UP$. By using a curve fitting, we derive an equation that models the effective throughput. The dashed lines in

Figure 4.5 (a) illustrate the derived equation for different network upload throughputs. The derived equation matches the Equation (4.3) from the proposed analytical model. The constant that corresponds to the time to set up the connection for our setup, *T.SC*, is 0.36 seconds.

Figure 4.5 (b) shows the effective energy efficiencies for uncompressed uploads over the encrypted WLAN channel as a function of the file size and the network connection energy efficiency, *EE.UP*. The plots show that the effective energy efficiency saturates for the larger files, reaching the network connection energy efficiency, i.e., *EE.UUP=EE.UP*. By using a curve fitting, we derive an equation that models the effective energy efficiency. The dashed lines in Figure 4.5 (b) illustrate the derived equations for different network upload throughputs. The derived equations match the Equation (4.4) from the proposed analytical model with two constants corresponding to *ET.SC* and *EE.UP*. For the setup used in our experiment *ET.SC* is 0.17 Joules.



(a)

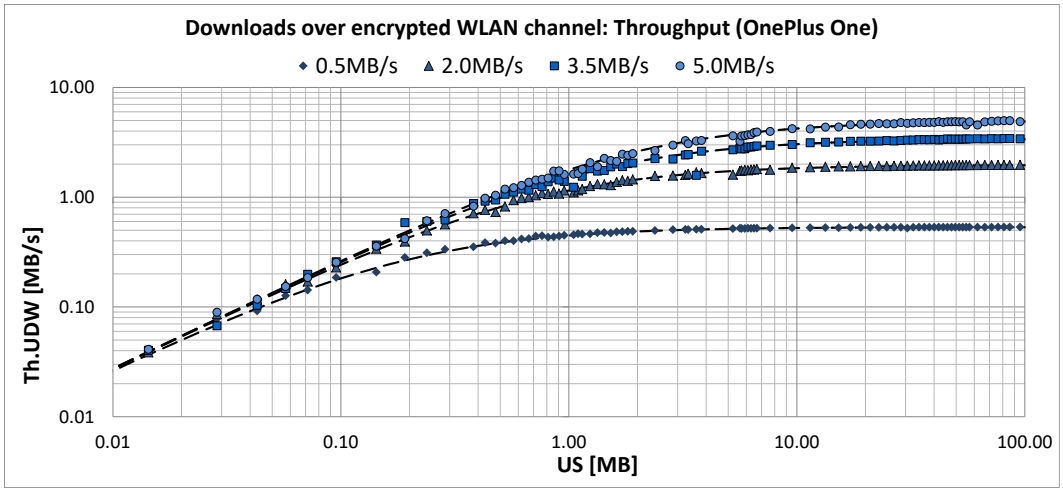


(b)

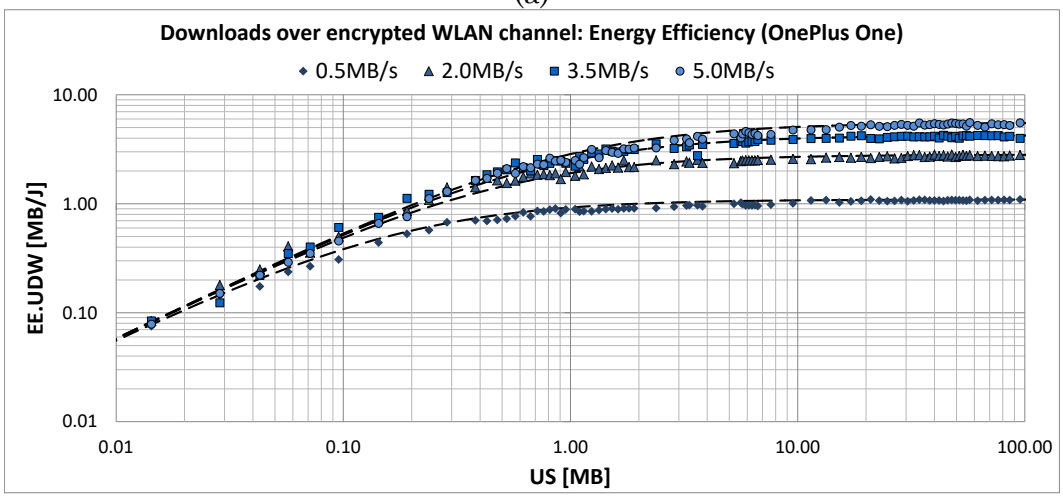
Figure 4.5 Uploads over encrypted WLAN channel (OnePlus One): measured throughputs (a) and energy efficiencies (b)

Figure 4.6 (a) and Figure 4.6 (b) show the measured effective throughputs and energy efficiencies for uncompressed file downloads over the encrypted WLAN channel for different network throughputs, respectively, as a function of the file size. The results of the download experiments confirm the correctness of the proposed analytical models for the effective throughput and energy efficiency for uncompressed

file downloads. Derived constants for *T.SC* and *ET.SC* match those derived from the upload experiments.



(a)

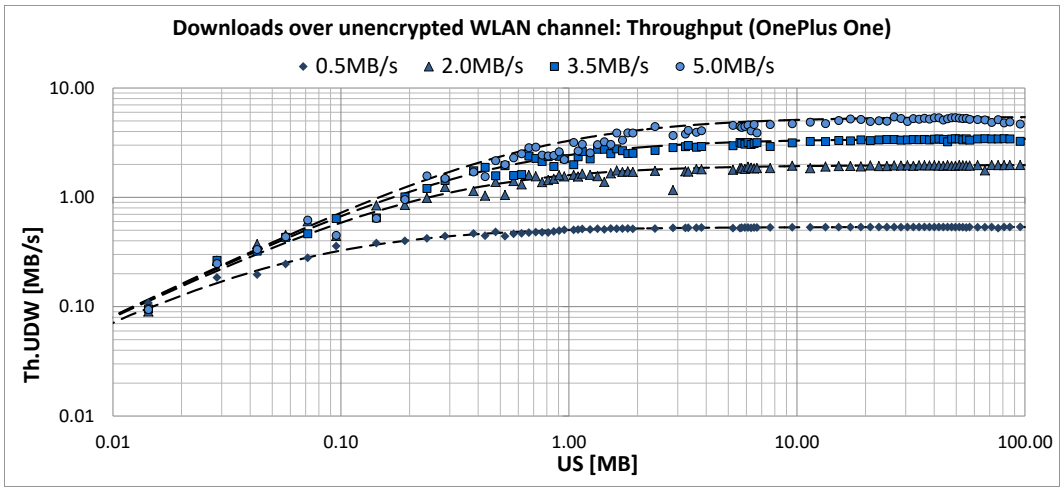


(b)

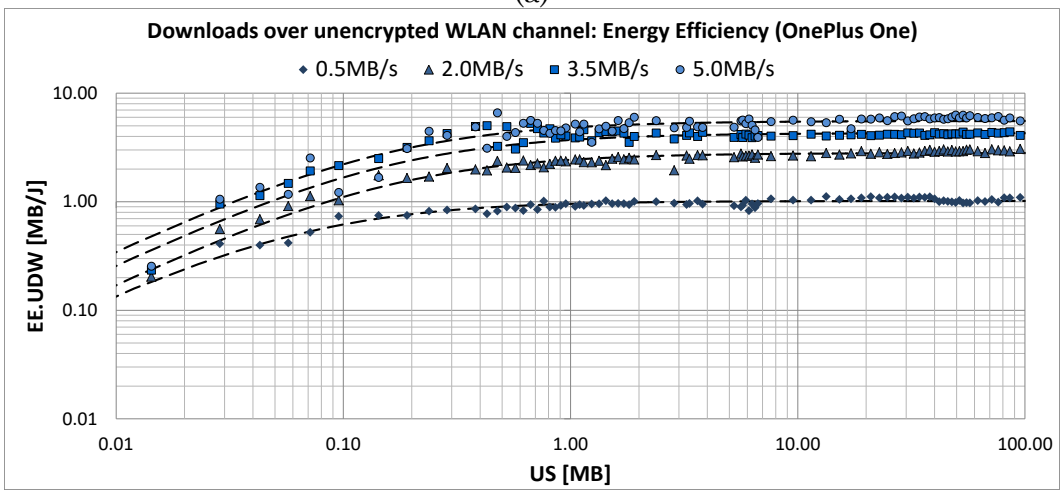
Figure 4.6 Downloads over encrypted WLAN channel (OnePlus One): measured throughputs (a) and energy efficiencies (b)

Figure 4.7 (a) and Figure 4.7 (b) show the measured effective throughputs and energy efficiencies for a set of download experiments carried over the unen-

crypted WLAN channel. The derived constants for *T.SC* and *ET.SC* are smaller than those for data transfers over the encrypted WLAN channel. *T.SC* is 0.12 seconds and *ET.*SC ranges from 0.03 Joules (for high network throughput) to 0.06 Joules (for low network throughput), respectively. As a result, the effective throughputs and energy efficiencies reach the saturation for smaller file sizes than for downloads over the encrypted WLAN channel. For the 5 MB/s network, the effective throughputs reaches 90% of network throughput at 6 MB on the unencrypted WLAN channel, and at 20 MB on the encrypted WLAN channel. The dashed lines in Figure 4.7 (a) and Figure 4.7 (b) illustrate the derived equations for different network upload and download throughputs.



(a)



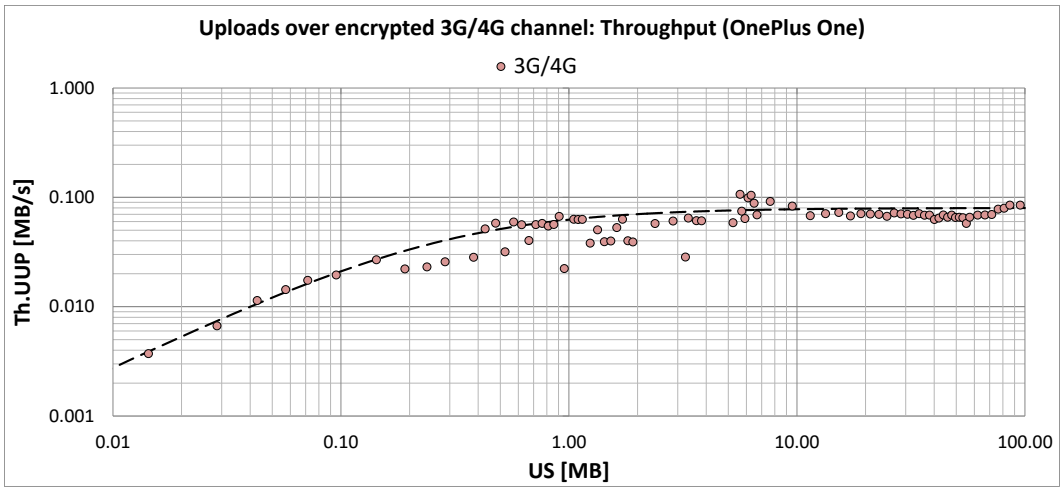
(b)

Figure 4.7 Downloads over unencrypted WLAN channel (OnePlus One): measured throughputs (a) and energy efficiencies (b)

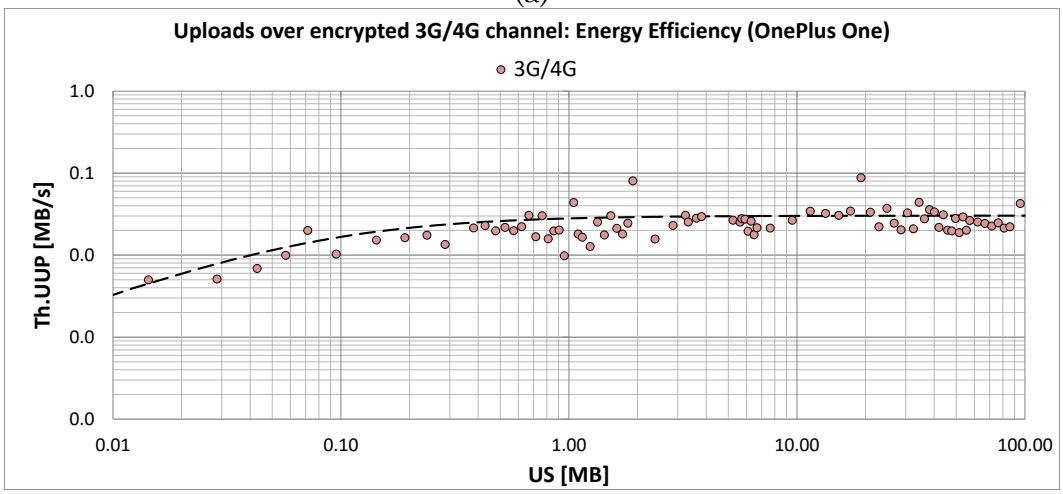
Mobile device (3G/4G). Figure 4.8 (a) and Figure 4.9 (a) show the measured effective throughputs for uncompressed uploads and downloads over the encrypted 3G/4G channel. The plots show that the achievable network throughputs are much smaller compared to WLAN transfers, reaching only 0.08 MB/s for upload and 0.5 MB/s for download. The derived constants for $T.SC$ are significantly larger than

ones for data transfers over the encrypted and unencrypted WLAN channel and are 3 to 3.5 seconds for both upload and download.

Figure 4.8 (b) and Figure 4.9 (b) show the measured effective energy efficiency for uncompressed uploads and downloads over the encrypted 3G/4G channel. The plots show that the achievable network energy efficiency are much smaller compared to WLAN transfers, reaching only 0.04 MB/J for upload and 0.4 MB/J for download. The derived constants for *ET.SC* are also significantly larger than ones for data transfers over the encrypted and unencrypted WLAN channel and are 4 to 5 Joules for both upload and download. The dashed lines in Figure 4.8 and Figure 4.9 illustrate the derived equations for different network upload and download throughputs and energy efficiencies.

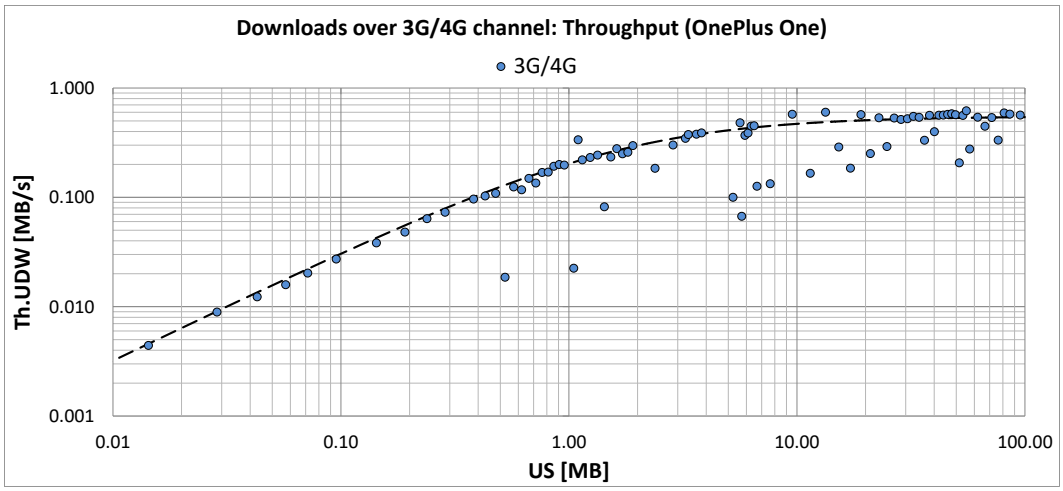


(a)

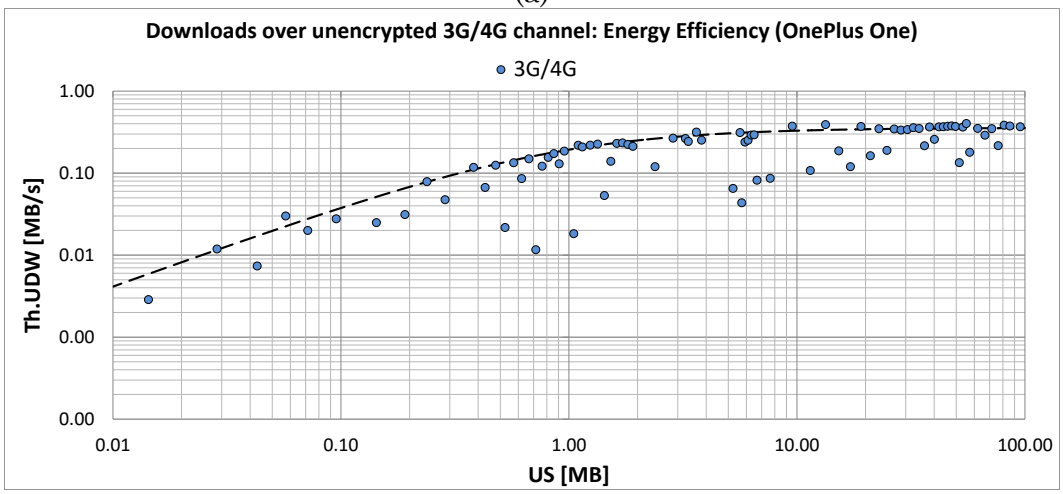


(b)

Figure 4.8 Uploads over encrypted 3G/4G channel (OnePlus One): measured throughputs (a) and energy efficiencies (b)



(a)



(b)

Figure 4.9 Downloads over encrypted 3G/4G channel (OnePlus One): measured throughputs (a) and energy efficiencies (b)

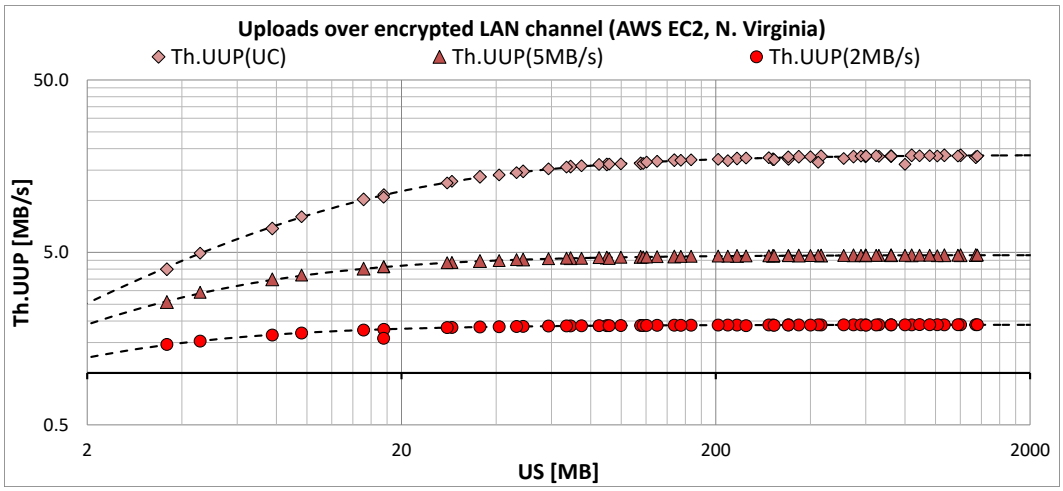
Workstation and the local server (LAN). The same set of experiments is repeated using the Dell PowerEdge T110 II workstation as an edge device. Due to faster processor and LAN network interface, the parameters *T.SC* and *ET.SC* are smaller than in the experiments conducted on OnePlus One. *T.SC* and *ET.SC* are 0.21 seconds and 0.11 Joules for the encrypted channel, and 0.015 seconds and 0.011 Joules for the unencrypted channel.

Workstation and the Cloud (LAN). Performing experiments with transfers to and from the cloud, instead of the local server, adds additional variable that affects network parameters. The distance to the cloud instance increases the time and energy to set up the connection, *T.SC [ET.SC]*, and lowers the network throughput and energy efficiency, *Th.UP [EE.UP]* and *Th.DW [EE.DW]*.

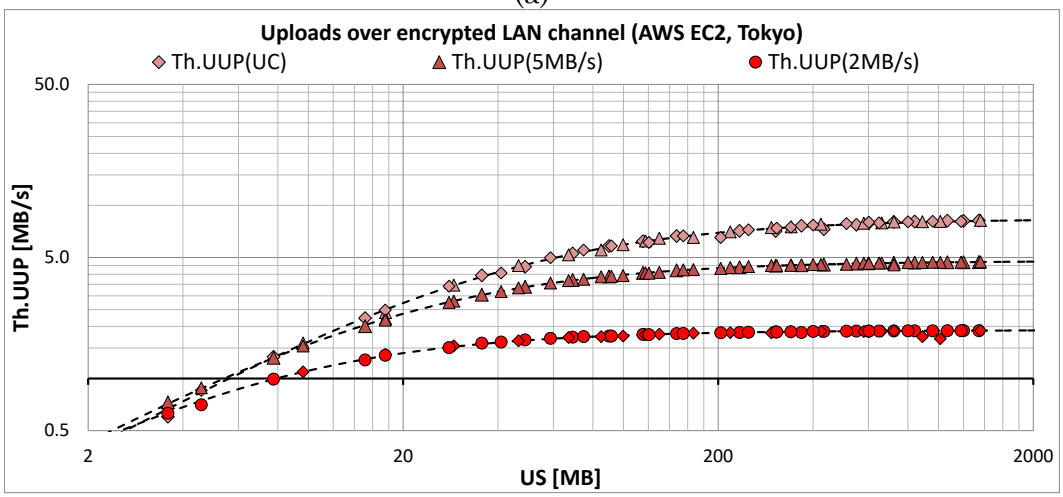
Figure 4.10 shows the measured effective throughputs for uncompressed uploads to the North Virginia and Tokyo cloud instances over the encrypted LAN channel. For uploads to the North Virginia instance, the network throughput for the uncapped network is 18.40 MB/s. The derived time to set up a network connection ranges from 0.68 seconds (on the uncapped network) to 0.58 seconds (on the 2 MB/s network). For uncompressed uploads to Tokyo instance, the network throughput for the uncapped network is 8.38 MB/s. The derived time to set up a network connection has increased due to increased distance from the requesting client and range from 4.9 seconds (on the uncapped network) to 3.75 (on the 2 MB/s network). Increased time to set up a network connection results in much slower saturation of the effective throughput, which reaches 90% of network throughput on the uncapped network at 367 MB, instead of 120 MB as with uploads to North Virginia.

Figure 4.11 shows the measured effective throughputs for uncompressed downloads to North Virginia and Tokyo cloud instances over encrypted LAN channel. For downloads from North Virginia instance, the effective throughput for the uncapped network is 80.61 MB/s. The derived time to set up a network connection ranges from 0.65 seconds (on the uncapped network) to 0.56 seconds (on the 2 MB/s network). For downloads from Tokyo instance, the effective throughput for uncapped download is smaller than for North Virginia instance, reaching 10.45 MB/s. The derived time to set up a network connection has increased due to increased distance,

and ranges from 4.4 seconds (on the uncapped network) to 3.5 (on the 2 MB/s network). The effective throughput saturates much slower, reaching 90% of network throughput on the uncapped network at 400 MB, instead of 250 MB as with uploads to North Virginia.

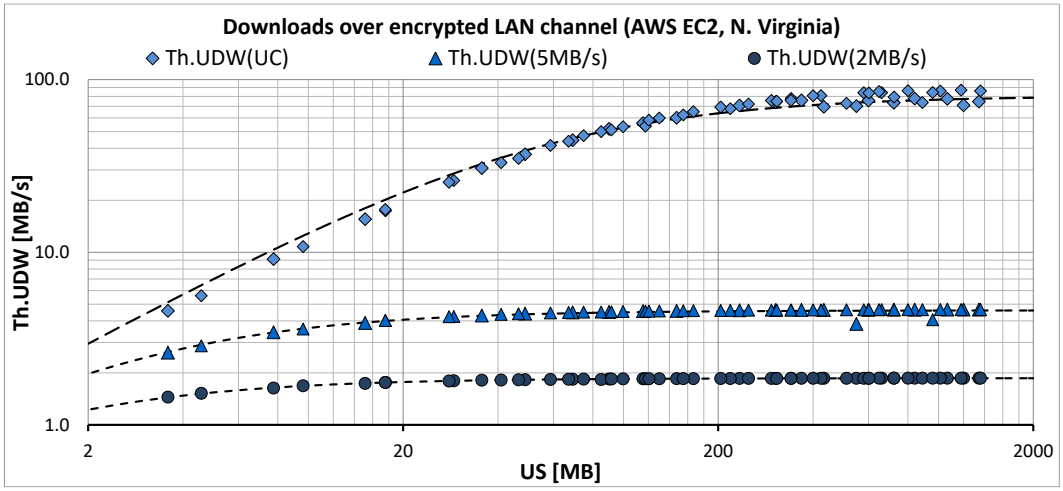


(a)

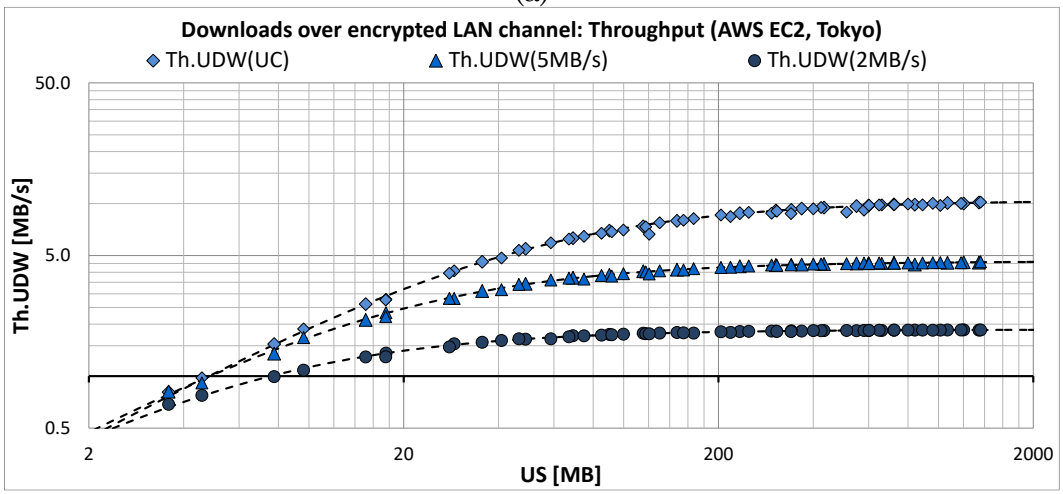


(b)

Figure 4.10 Uploads over encrypted LAN channel (Dell PowerEdge T110 II): measured throughputs: North Verginia (a) and Tokyo (b)



(a)



(b)

Figure 4.11 Downloads over encrypted LAN channel (Dell PowerEdge T110 II): measured throughputs: North Virginia (a) and Tokyo (b)

4.2.1.3 Network Connection Characterization

The experimental verification of the models for the effective throughput and energy efficiency requires a series of uploads and downloads of data files of different sizes. However, such an approach is not practical in real conditions because it takes

considerable time and requires instrumentation of smartphone for performing energy measurements. Here we describe a practical approach for deriving unknown performance and energy network parameters using the verified analytical model and a limited number of experiments. Specifically, we describe practical experiments that derive the following parameters:

- The upload and download network throughputs, *Th.UP [Th.DW]*,
- The network upload and download energy efficiencies, *EE.UP [EE.DW]*,
- The time and energy spent to set up the network connection, *T.SC [ET.SC]*

The proposed method for deriving the network parameters involves performing a two file upload or download test. Two files of different sizes are selected to be uploaded or downloaded over a network connection with unknown parameters. The total transfer time is measured and used to calculate the effective throughput. These measured quantities are then used within the models to derive the unknown performance parameters, *Th.UP [Th.DW]* and *T.SC*. The derived performance network parameters are then used to estimate energy network parameters, *EE.UP [EE.DW]* and *ET.SC*, using known device characteristics (device battery voltage, device idle current, and the maximum active peak currents of data file transfers) and Equations (4.11), (4.12) and (4.13).

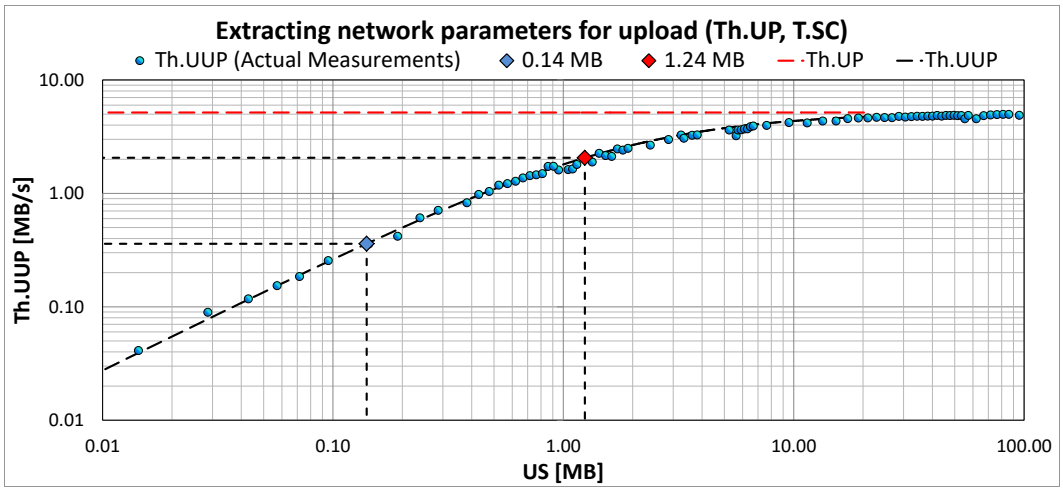
To demonstrate the derivation of performance and energy network parameters, we consider file uploads over an *ssh* network connection that utilizes the smartphone's WLAN interface. The goal is to determine the *Th.UP* and *T.SC* and then estimate *EE.UP* and *ET.SC*. We select two test files with sizes *US(s)*=0.14 MB

and *US(l)*=1.24 MB. The measured effective throughputs are *Th.UUP(s)*=0.36 MB/s for the 0.14 MB file and *Th.UUP(l)*=2.06 MB/s for the 1.24 MB file. Next, by replacing the file sizes and the measured effective throughputs in Equation (4.14) we get two equations with two unknowns, *T.SC* and *Th.UP*. By solving the system of linear equations, shown in Equation (4.15), we derive values of 5.167 MB/s and 0.362 seconds for *Th.UP* and *T.SC*, respectively. Using Equations (4.11) and (4.13), energy parameters (*EE.UP* and *ET.SC*) are estimated to be 5.868 MB/J and 0.16 Joules, respectively.

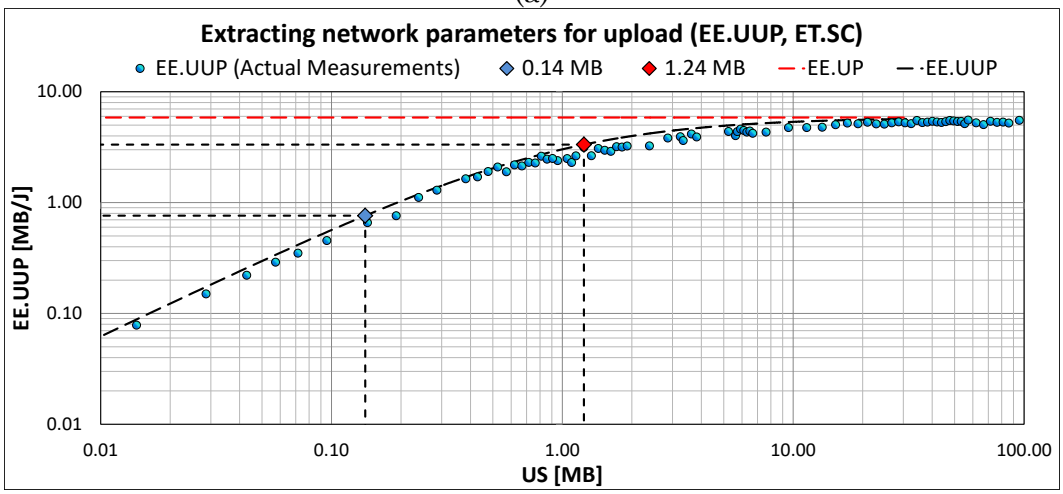
$$Th.UP = \frac{Th.UUP}{1 - Th.UUP \cdot T.SC/US} \tag{4.14}$$

$$\begin{aligned} Th.UP - 2.57 \cdot Th.UP \cdot T.SC &= 0.36 \\ Th.UP - 1.66 \cdot Th.UP \cdot T.SC &= 2.06 \end{aligned} \tag{4.15}$$

Figure 4.12 (a) and (b) illustrates the proposed method for characterizing network connection with performance and energy-based metrics. The measured upload throughputs and the estimated energy efficiencies for two selected files are marked with a blue and a red diamond in both figures respectively. By deriving *Th.UP* and *T.SC*, and estimating *EE.UP* and *ET.SC,* the models from Equation (4.3) and (4.7) are plotted using a black dashed-dot curve. The actual measurements of the effective upload throughputs (a) and effective upload energy efficiencies (b) performed during the verification phase are shown as blue circles. A visual inspection shows that the model with parameters extracted (performance) and estimated (energy) by just two measurements matches the actual measurements of throughputs and energy efficiencies performed during the verification phases.



(a)



(b)

Figure 4.12 Extracting network parameters for uploads: throughput (a) and estimated energy efficiency (b)

4.2.2 Modeling Compressed File Transfers

A compressed upload of a data file to the cloud and a compressed download from the cloud can be performed in two ways, sequentially or with the use of piping. In case of a sequential upload, the data file is first compressed locally on the edge device and then compressed file is transferred to the cloud, with no overlap between these two tasks. Similarly, in case of a sequential download, the compressed data file

is first downloaded on the edge device and then decompressed. In case of piped file transfers, the file (de)compression times are partially or completely hidden by the time to set up the network connection and the file transmission time. When the server does not have the requested compressed file, the uncompressed data file is first compressed on the server, and then the compressed file is downloaded on the edge device sequentially or with piping.

This section is organized as follows. Subsections 4.2.2.1 and 4.2.2.2 introduce upper and lower limits for the throughput and energy efficiency. The final analytical models for the effective throughput and energy efficiency for piped file transfers are introduced in Subsection 4.2.2.3. Subsection 4.2.2.4 presents a practical way of estimating energy efficiency from the performance predication data. Finally, Subsection 4.2.2.5 describes how the framework utilizes the models when performing optimized uploads and downloads.

4.2.2.1 Throughput Limits for Compressed File Transfers

Compressed uploads. The maximum compressed upload time shown in Equation (4.16), $T.CUP.max$, includes the time to perform the local compression of the file on the edge device, the time to set up a network connection, $T.SC$, and the time to transfer the compressed file, $T.CUP'$. The time to transfer the compressed file can be calculated as the compressed file size, which is US/CR, where CR is the compression ratio, divided by the network connection upload throughput $Th.UP$. Instead of using the time to perform local compression on the edge device, $T.C$, we can use the local compression throughput, $Th.C$, defined as the uncompressed file size, US, divided by the time to perform a local compression, $T.C$. This "higher is better" metric captures ability of the edge device to perform local compression fast. The min-

imum compressed upload time shown in Equation (4.17), *T.CUP.min*, includes the time to set up the network connection, *T.SC*, and the time to transfer the compressed file, *T.CUP'*.

$$T.CUP.max = T.C + T.SC + T.CUP' \tag{4.16}$$

$$T.CUP.min = T.SC + T.CUP' \tag{4.17}$$

$$Th.CUP.min = \frac{US}{T.CUP.min} = \frac{CR \cdot Th.UP}{1 + CR \cdot Th.UP \cdot (\frac{1}{Th.C} + \frac{T.SC}{US})} \tag{4.18}$$

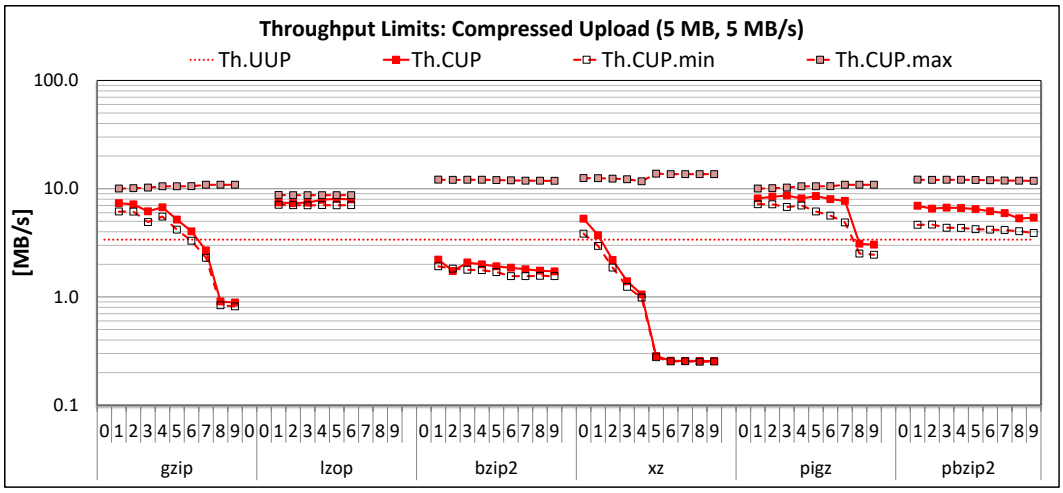
$$Th.CUP.max = \frac{US}{T.CUP.max} = \frac{CR \cdot Th.UP}{1 + CR \cdot Th.UP \cdot T.SC/US} \tag{4.19}$$

The minimum effective compressed upload throughput, *Th.CUP.min*, is calculated as the uncompressed file size in megabytes, *US*, divided by the maximum time to perform compressed upload, *T.CUP.max*. The maximum effective compressed upload throughput, *Th.CUP.max*, is calculated as the uncompressed file size in megabytes, *US*, divided by the minimum time to perform compressed upload *T.CUP.min*. The final expressions in Equations (4.18) and (4.19) show the boundaries for the compressed upload throughputs as a function of the network parameters, *Th.UP* and *T.SC*, the file size, *US*, the compression ratio, *CR*, and the local compression throughput, *Th.C*. From these expressions, we can analytically estimate the impact of changes in these parameters to the effective throughputs. For example, the highest compressed upload throughput that can be achieved approaches the product of the compression ratio and the network connection upload throughput, which is possible in devices where local compression throughputs exceeds the net-

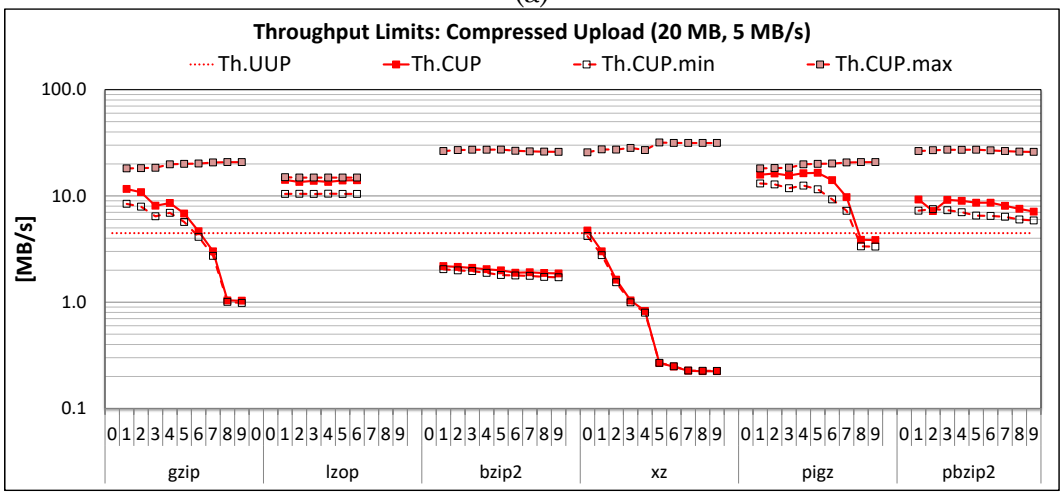
work upload throughput and when the size of a transferred file is sufficiently large so that transfer time dwarfs the network connection setup time.

Figure 4.13 illustrates the estimated minimum and maximum throughputs, *Th.CUP.min* and *Th.CUP.max*, respectively, as well as the measured compressed upload throughput, *Th.CUP*, for different modes of compressed upload for three files with file sizes of 5 MB (a), 20 MB (b), and 100 MB (c). The three files are selected from the dataset containing Zephyr Technologies BioHarness 3 chest belt raw samples in CSV format, including acceleration, breathing, and ECG data recorded during various activities of a subject. The 5 MB file contains ECG samples recorded during a 7-minute scientific workout [85], whereas 20 MB and 100 MB files contain breathing and acceleration data from a subject's 6-hour sleep, respectively. The measurements are performed on the OnePlus One smartphone with a 5 MB/s WLAN network. The measured compressed upload throughput is between the predicted minimum and maximum throughputs. For example, the estimated lower and upper limits for the compression throughput of *gzip* with -1 are 6.15 MB/s and 10 MB/s for the 5 MB file, 8.42 MB/s and 18.126 MB/s for the 20 MB file, and 11.79 MB/s and 31.09 MB/s for the 100 MB file, while the measured compression throughputs are 7.35, 11.62 and 16.18 MB/s, respectively. In contrast, the estimated bounds for *bzip2* with -1 are 1.92 MB/s and 60.57 MB/s for the 5 MB file, 2.04 MB/s and 26.43 MB/s for the 20 MB file, and 1.92 MB/s and 12.11 MB/s for the 100 MB file, while the measured compression throughputs are 2.21, 2.18 and 1.95 MB/s, respectively. In cases when the local compression throughput falls below the network connection upload throughput, $Th.C << Th.UP$, the effective compressed upload throughput is closer to the minimum throughput (e.g., for *xz*). In cases when $Th.C >> Th.UP$, the

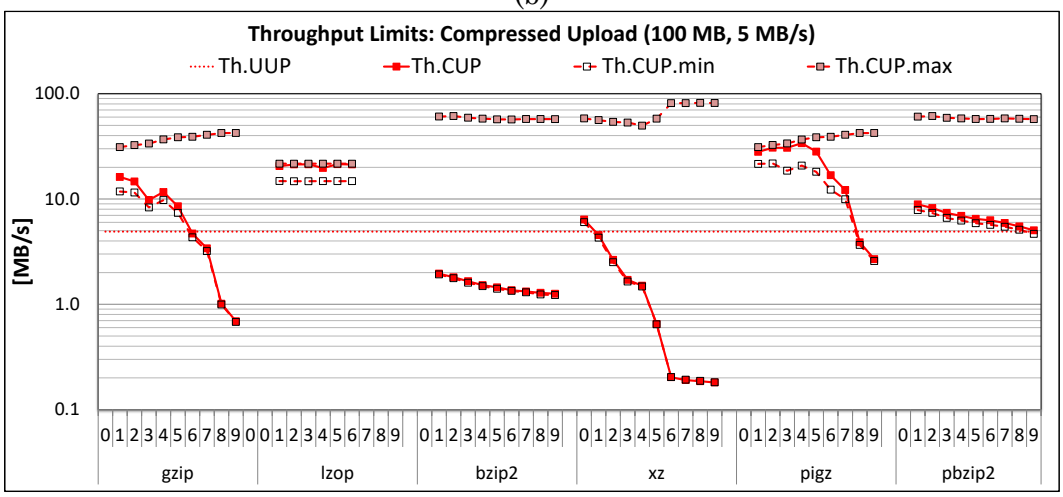
effective compressed upload throughput is closer to the expected maximum throughput (e.g, for *lzop*).



(a)



(b)



(c)

Figure 4.13 Throughput limits: compressed uploads for 5 MB (a), 20 MB (b), and 100 MB (c) files

Compressed downloads. Assuming the compressed file is available on the server, the maximum compressed download time shown in Equation (4.20), $T.CDW.max$, includes the time to set up the network connection, $T.SC$, the time to transfer the compressed file, $T.CDW'$, and the time to perform the decompression of the received file on the edge device. The time to transfer the compressed file can be calculated as the compressed file size, US/CR, divided by the network connection download throughput $Th.DW$. The time to perform decompression on the edge device, $T.D$, can be used to determine the local decompression throughput, $Th.D$, which is defined as the uncompressed file size, US, divided by the time to perform decompression. This metric thus captures the edge device's ability to effectively perform decompression. The minimum download time shown in Equation (4.21), $T.CDW.min$, includes the time to set up the network connection, $T.SC$, and the time to transfer the compressed file, $T.CDW'$.

$$T.CDW.max = T.D + T.SC + T.CDW' \tag{4.20}$$

$$T.CDW.min = T.SC + T.CDW' \tag{4.21}$$

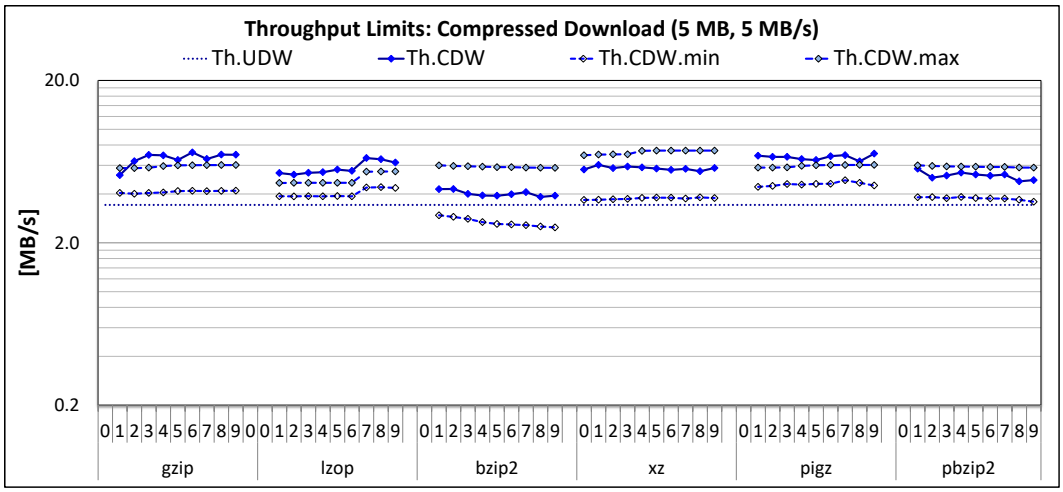
$$Th.CDW.min = \frac{US}{T.CDW.min} = \frac{CR \cdot Th.DW}{1 + CR \cdot Th.DW \cdot (\frac{1}{Th.D} + \frac{T.SC}{US})} \tag{4.22}$$

$$Th.CDW.max = \frac{US}{T.CDW.max} = \frac{CR \cdot Th.DW}{1 + CR \cdot Th.DW \cdot T.SC/US} \tag{4.23}$$

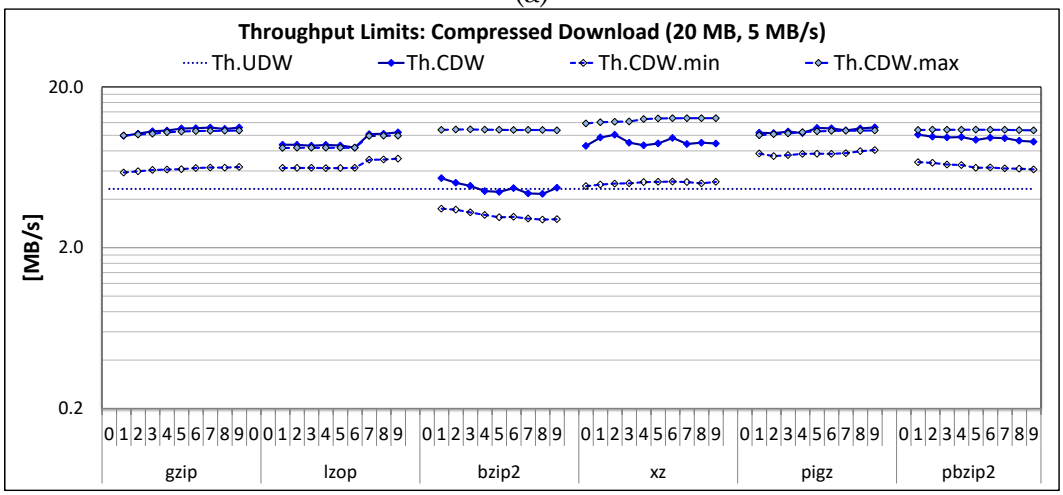
The minimum effective compressed download throughput, $Th.CDW.min$, is calculated as the uncompressed file size in megabytes, US, divided by the maximum time to perform compressed upload, $T.CDW.max$. The maximum effective compressed download throughput, $Th.CDW.max$, is calculated as the uncompressed file

size in megabytes, *US*, divided by the minimum time to perform the compressed download, *T.CDW.min*. The final expressions in Equations (4.22) and (4.23) show the boundaries for the compressed download throughputs as a function of the network parameters, the file size, the compression ratio, and the local decompression throughput.

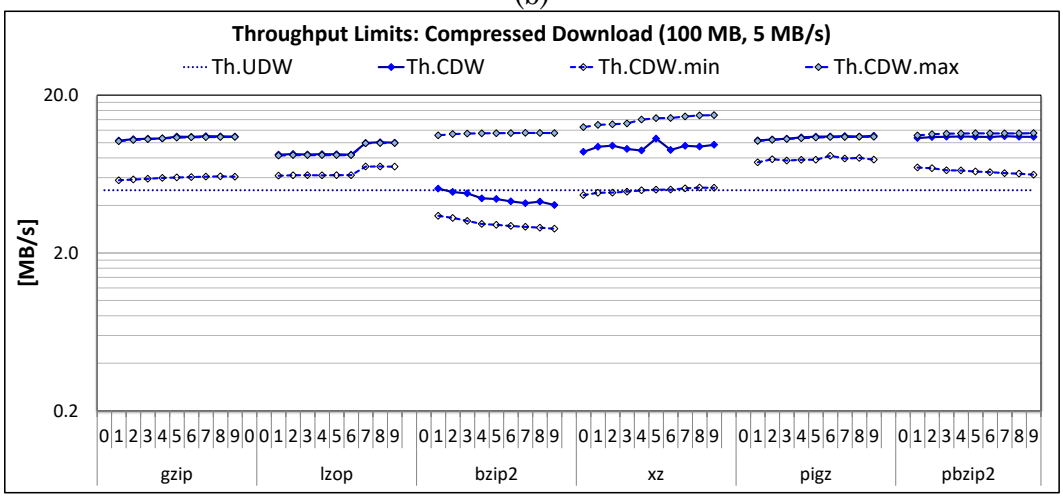
Figure 4.14 illustrates the estimated throughput boundaries and the measured compressed download throughput for different modes of compressed download for three files with file sizes of 5 MB (a), 20 MB (b), and 100 MB (c). The three files are selected from the dataset containing the most popular Android applications, which were repackaged into uncompressed archive files (tar). The 5 MB file is a dictionary application, the 20 MB file is an application for online mobile photo-sharing service, and the 100 MB file is one of the popular Android games. The measurements are performed on the OnePlus One smartphone with a 5 MB/s WLAN network. The measured compressed download throughput is between the predicted minimum and maximum throughputs. For example, the estimated lower and upper boundaries for the decompression throughput of *gzip* with -9 are 5.34 MB/s and 7.02 MB/s for the 5 MB file, 7.88 MB/s and 11.32 MB/s for the 20 MB file, and 7.42 MB/s and 11.02 MB/s for the 100 MB file, while the measured compression throughputs are 6.89, 11.0 and 10.84 MB/s. The utilities with high local decompression throughputs achieve the effective download throughputs close to the upper boundaries when downloading large files (e.g., *gzip*, *lzop* and *pigz* for all compression levels).



(a)



(b)



(c)

Figure 4.14 Throughput limits: compressed downloads for 5 MB (a), 20 MB (b), and 100 MB (c) files

Compressed downloads with on-demand compression. When the compressed file is not available on the server, the maximum compressed download time shown in Equation (4.24), *T.CDW.max(s)*, includes the time to set up the network connection, *T.SC*, the time to compress the file on the server, *T.C(s)*, the time to transfer the compressed file, *T.CDW'*, and the time to perform the decompression of the received file on the edge device, *T.D*. The time to perform compression on the server, *T.C(s)*, can be used to determine the server's compression throughput, *Th.C(s)*, which is defined as the uncompressed file size, *US*, divided by the time to perform compression. The minimum download time, *T.CDW.min,* and maximum download throughput, *Th.CDW.max*, remain the same as defined in Equations (4.21) and (4.23), respectively.

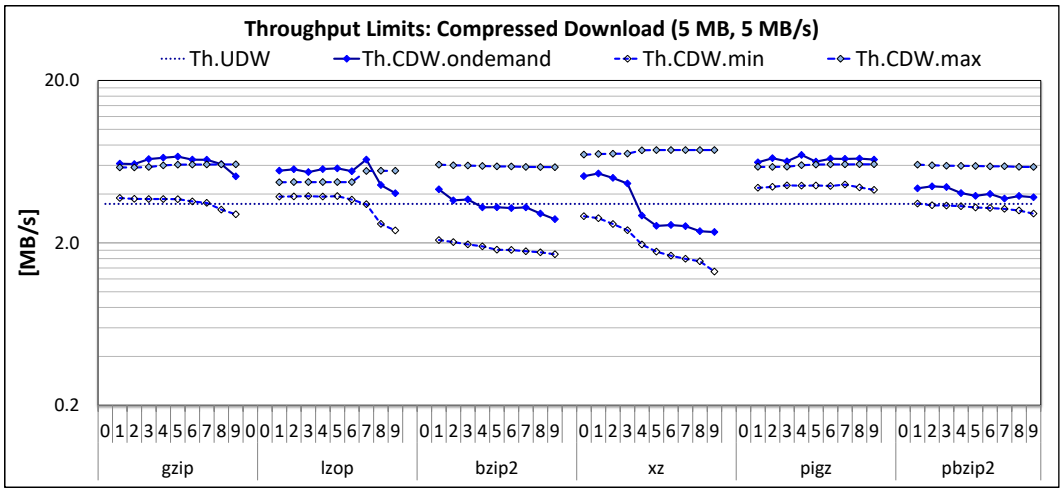
$$T.CDW.\max(s) = T.D + T.SC + T.C(s) + T.CDW' \tag{4.24}$$

$$Th.CDW.\min(s) = \frac{US}{T.CDW.\max(s)} = \frac{CR \cdot Th.DW}{1 + CR \cdot Th.DW \cdot (\frac{1}{Th.D} + \frac{1}{Th.C(s)} + \frac{T.SC}{US})} \tag{4.25}$$

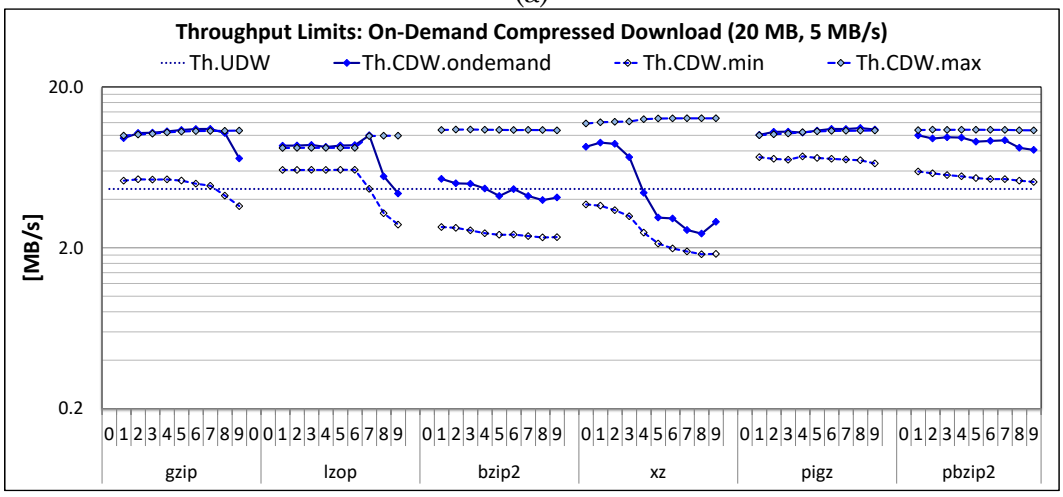
The minimum effective compressed download throughput, *Th.CDW.min(s)*, is calculated as the uncompressed file size in megabytes, *US*, divided by the maximum time to perform compressed upload, *T.CDW.max(s)*. The final expressions in Equations (4.25) and (4.23) show the boundaries for the compressed download throughputs as a function of the network parameters, the file size, the compression ratio, the local compression throughput on the server, and the local decompression throughput on the edge device.

Figure 4.15 illustrates the estimated throughput boundaries and the measured compressed download throughput for different modes of compressed download

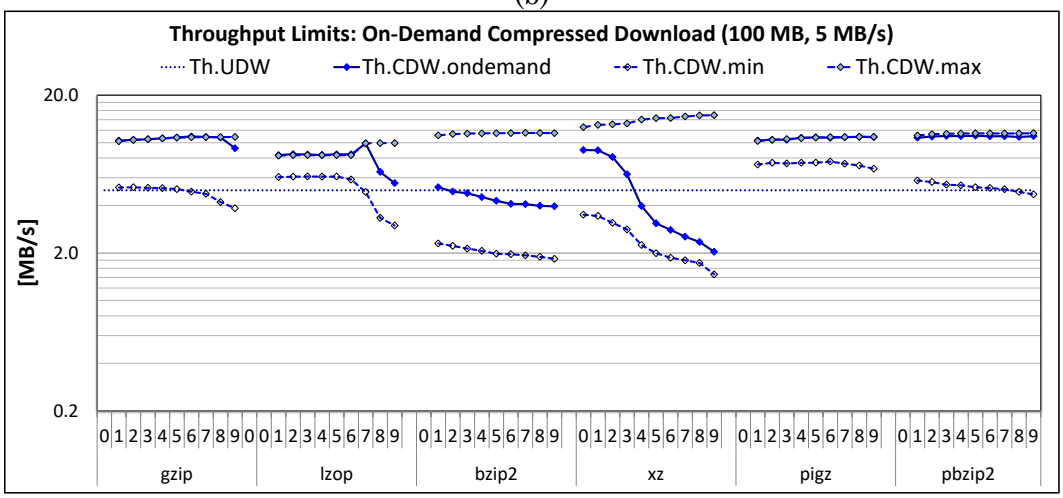
with on-demand compression for three files with file sizes of 5 MB (a), 20 MB (b), and 100 MB (c). The measurements are performed on the OnePlus One smartphone with a 5 MB/s WLAN network. The measured compressed download throughput is between the predicted minimum and maximum throughputs. For example, the estimated lower and upper boundaries for the decompression throughput of *gzip* with -9 are 3.52 MB/s and 7.02 MB/s for the 5 MB file, 4.09 MB/s and 11.32 MB/s for the 20 MB file, and 4.32 MB/s and 11.02 MB/s for the 100 MB file, while the measured compression throughputs are 5.13, 7.34 and 9.23 MB/s, respectively. The utilities with high local decompression throughputs and high remote compression throughputs achieve the effective download throughputs close to the upper boundaries when downloading large files (e.g., *gzip*, *lzop* and *pigz* for all compression levels). Additionally, due to the server-side compression overhead, the minimum throughputs and actual measured throughputs are lower than before, especially for smaller files. For example, the measured throughputs with on-demand compression are lower by 25.5%, 16.1%, and 14.9% for the 5 MB, 20 MB, and 100 MB files, respectively, than the download throughputs with pre-compressed files.



(a)



(b)



(c)

Figure 4.15 Throughput limits: compressed downloads with on-demand compression for 5 MB (a), 20 MB (b), and 100 MB(c) files

4.2.2.2 Energy Efficiency Limits for Compressed File Transfers

Compressed uploads. The maximum compressed upload energy shown in Equation (4.26), *ET.CUP.max*, includes the energy to perform the local compression of the file on the edge device, the energy to set up the network connection, *ET.SC*, and the energy to transfer the compressed file, *ET.CUP'*. The energy to transfer the compressed file can be calculated as the compressed file size, *US/CR*, divided by the network connection for upload energy efficiency, *EE.UP*. Instead of using the energy to perform local compression on the edge device, *ET.C*, we can be used the local compression energy efficiency, *EE.C*, defined as the uncompressed file size, *US*, divided by the energy to perform a local compression, *ET.C*. This metric captures the edge device's ability to perform compression with the least amount of energy. The minimum compressed upload energy shown in Equation (4.27), *ET.CUP.min*, includes the energy overhead to perform the local compression of the file on the edge device, *ET.C(0)*, the energy to set up the network connection, *ET.SC*, and the energy to transfer the compressed file of size, *ET.CUP'*. The energy overhead, *ET.C(0)*, excludes the energy needed to run the platform when idle.

$$ET.CUP.max = ET.C + ET.SC + ET.CUP' \tag{4.26}$$

$$ET.CUP.min = ET.C(0) + ET.SC + ET.CUP' \tag{4.27}$$

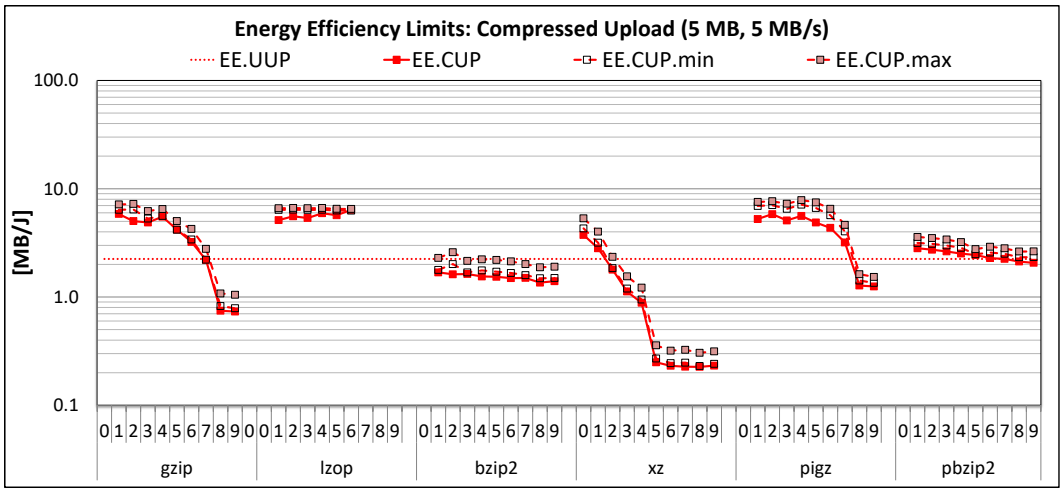
$$EE.CUP.min = \frac{US}{ET.CUP.max} = \frac{CR \cdot EE.UP}{1 + CR \cdot EE.UP \cdot (\frac{1}{EE.C} + \frac{ET.SC}{US})} \tag{4.28}$$

$$EE.CUP.max = \frac{US}{ET.CUP.min} = \frac{CR \cdot Th.UP}{1 + CR \cdot EE.UP \cdot (\frac{1}{EE.C(0)} + \frac{ET.SC}{US})} \tag{4.29}$$

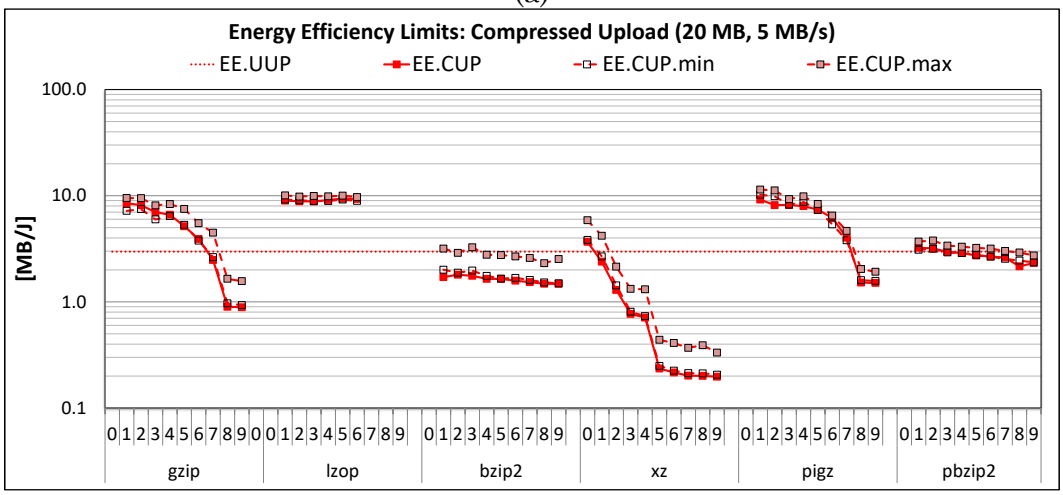
The minimum effective compressed upload energy efficiency, *EE.CUP.min*, is calculated as the uncompressed file size in megabytes, *US*, divided by the maximum energy to perform compressed upload, *ET.CUP.max*. The maximum effective compressed upload energy efficiency, *EE.CUP.max*, is calculated as the uncompressed file size in megabytes, *US*, divided by the minimum energy to perform compressed upload, *ET.CUP.min*. The final expressions in Equations (4.28) and (4.29) show the boundaries for the compressed upload energy efficiencies as a function of the network parameters, *EE.UP*, *ET.SC*, the file size, *US*, the compression ratio, *CR*, and the local compression energy efficiency, *EE.C*.

Figure 4.16 illustrate the estimated minimum and maximum energy efficiencies, *EE.CUP.min* and *EE.CUP.max*, respectively, as well as the measured compressed upload energy efficiency, *EE.CUP*, for different modes of compressed upload for three files with file sizes of 5 MB (a), 20 MB (b), and 100 MB (c). The three files correspond to selected files for characterization of upload in Section 4.2.2.1. The measurements are performed on the OnePlus One smartphone with a 5 MB/s WLAN network. The measured compressed upload energy efficiency is between the predicted minimum and maximum energy efficiencies. For example, the estimated lower and upper limits for the compression energy efficiency of *gzip* -1 are 6.4 MB/J and 7.16 MB/J for the 5 MB file, 7.2 MB/J and 9.5 MB/J for the 20 MB file, and 9.45 MB/J and 12.7 MB/J for the 100 MB file, while the measured compression energy efficiencies are 5.85, 8.51 and 9.96 MB/J, respectively. In contrast, the estimated bounds for *bzip2* with -1 are 1.8 MB/J and 2.29 MB/s for the 5 MB file, 2 MB/s and 3.18 MB/s for the 20 MB file, and 1.73 MB/s and 2..6 MB/s for the 100 MB file, while the measured compression throughputs are 1.7, 1.71 and 1.69 MB/s, respectively. In

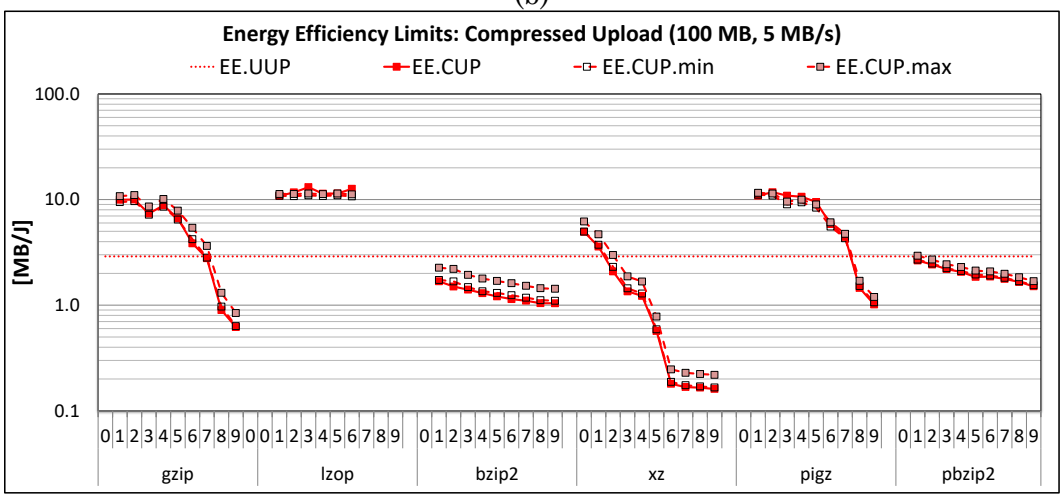
cases when the local compression energy efficiency falls below the network connection upload energy efficiency, $EE.C << EE.UP$, the effective compressed upload energy efficiency is closer to the minimum energy efficiency (e.g., for *xz*). In cases when $EE.C >> EE.UP$, the effective compressed upload energy efficiency is closer to the expected maximum energy efficiency (e.g., for *lzop*).



(a)



(b)



(c)

Figure 4.16 Energy efficiency limits: compressed uploads for 5 MB (a), 20 MB (b), 100 MB (c) files

Compressed downloads. Assuming the compressed file is available on the server, the maximum compressed download energy shown in Equation (4.30), *ET.CDW.max*, includes the energy to set up the network connection, *ET.SC*, the energy to transfer the compressed file, *ET.CDW′*, and the energy to perform the decompression of the received file on the edge device. The energy to transfer the compressed file can be calculated as the compressed file size, US/CR, divided by the network connection download energy efficiency *EE.DW*. The energy to perform decompression on the edge device, *ET.D*, can be used to determine the local decompression energy efficiency, *EE.D*, which is defined as the uncompressed file size, *US*, divided by the energy to perform decompression. This metric thus captures the edge device's ability to effectively perform decompression. The minimum download energy shown in Equation (4.31), *ET.CDW.min*, includes the energy to set up the network connection, *ET.SC*, the energy to transfer the compressed file, *ET.CDW′*, and the overhead energy to perform decompression, *ET.D(0)*.

$$ET.CDW.max = ET.D + ET.SC + ET.CDW' \tag{4.30}$$

$$ET.CDW.min = ET.D(0) + ET.SC + ET.CDW' \tag{4.31}$$

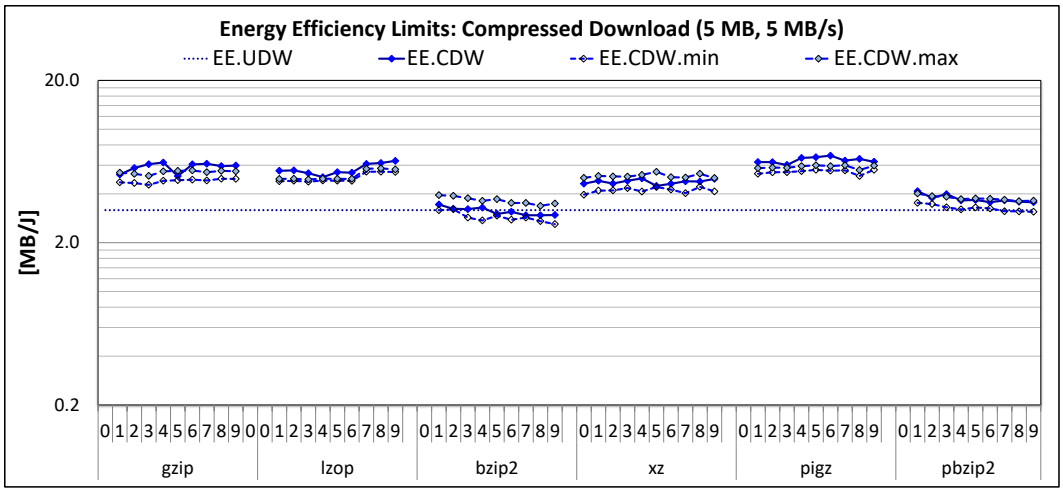
$$EE.CDW.min = \frac{US}{ET.CDW.max} = \frac{CR \cdot EE.DW}{1 + CR \cdot EE.DW \cdot (\frac{1}{EE.D} + \frac{ET.SC}{US})} \tag{4.32}$$

$$EE.CDW.max = \frac{US}{ET.CDW.min} = \frac{CR \cdot Th.DW}{1 + CR \cdot EE.DW \cdot (\frac{1}{EE.D(0)} + \frac{ET.SC}{US})} \tag{4.33}$$

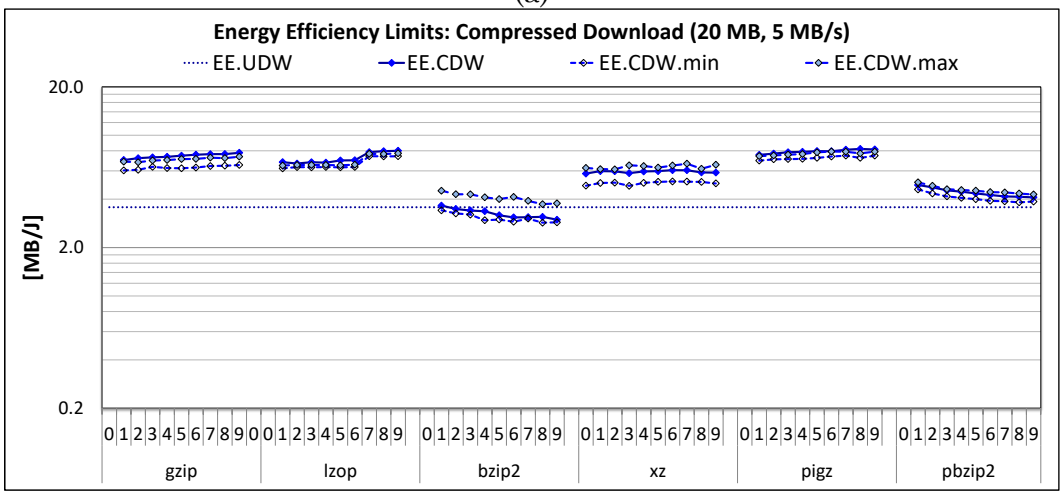
The minimum effective compressed download energy efficiency, *EE.CDW.min*, is calculated as the uncompressed file size in megabytes, *US*, divided by the maximum energy to perform the compressed download, *ET.CDW.max*. The

maximum effective compressed download energy efficiency, *EE.CDW.max*, is calculated as the uncompressed file size in megabytes, *US*, divided by the minimum energy to perform the compressed download, *ET.CDW.min*. The final expressions in Equations (4.32) and (4.33) show the boundaries for the compressed download energy efficiencies as a function of the network parameters, the file size, the compression ratio, and the local decompression energy efficiency.

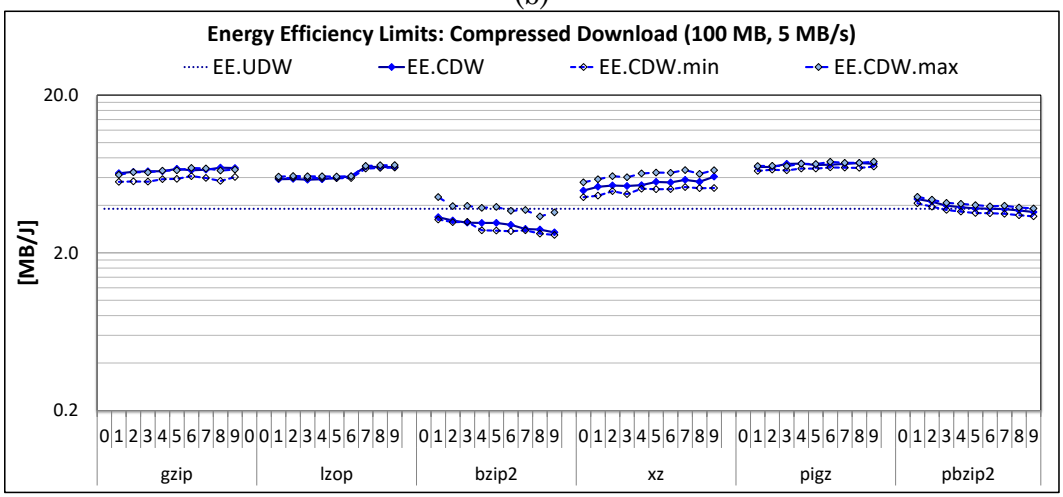
Figure 4.17 illustrates the estimated energy efficiency boundaries and the measured compressed download energy efficiency for different modes of compressed download for three files with file sizes of 5 MB (a), 20 MB (b), and 100 MB (c). The three files correspond to selected files for characterization of download in Section 4.2.2.1. The measurements are performed on the OnePlus One smartphone with a 5 MB/s WLAN network. The measured compressed download energy efficiency is between the predicted minimum and maximum energy efficiencies. For example, the estimated lower and upper boundaries for the decompression energy efficiency of *gzip* -9 are 4.95 MB/J and 5.5 MB/J for the 5 MB file, 6.53 MB/J and 7.36 MB/J for the 20 MB file, and 6.05 MB/J and 6.74 MB/J for the 100 MB file, while the measured compression energy efficiencies are 5.97, 7.78 and 6.89 MB/J. The utilities with high local decompression energy efficiencies achieve the effective download energy efficiencies close to the upper boundaries when downloading large files (e.g., *gzip*, *xz*, *pigz*, and *pbzip2* for all compression levels).



(a)



(b)



(c)

Figure 4.17 Energy efficiency limits: compressed downloads

for 5 MB (a), 20 MB (b), 100 MB (c) files

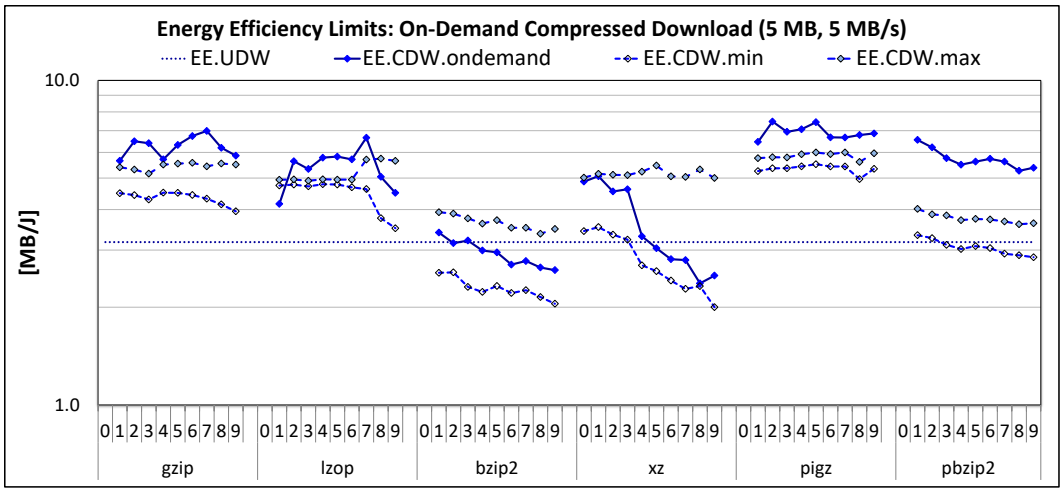
Compressed downloads (with on-demand compression). When the compressed file is not available on the server, the maximum compressed download energy shown in Equation (4.34), *ET.CDW.max(s)*, includes the energy to set up the network connection, *ET.SC*, the edge device idle energy while waiting for compression of the file on the on the server, $I_{idle} \cdot V_{BS} \cdot T.C(s)$, the energy to transfer the compressed file, *ET.CDW'*, and the energy to perform the decompression of the received file on the edge device, *ET.D*. The energy to perform compression on the server, *ET.C(s)*, can be used to determine the server's compression energy efficiency, *EE.C(s)*, which is defined as the uncompressed file size, *US*, divided by the time to perform compression. The minimum download energy, *ET.CDW.min,* and maximum download energy efficiency, *EE.CDW.max*, remain the same as defined in Equations (4.31) and (4.33), respectively.

$$ET.CDW.\max(s) = ET.D + ET.SC + ET.CDW' + V_{BS} \cdot I_{idle} \cdot T.C(s) \tag{4.34}$$

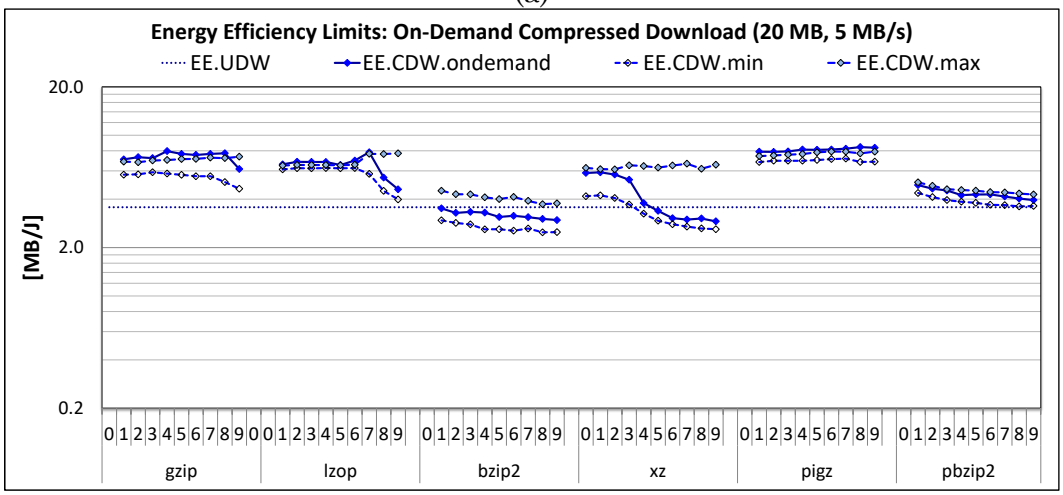
$$EE.CDW.\min(s) = \frac{US}{ET.CDW.\max(s)} = \frac{CR \cdot EE.DW}{1 + CR \cdot EE.DW \cdot \left(\frac{1}{EE.D} + \frac{1}{\frac{Th.C(s)}{V_{BS} \cdot I_{idle}}} + \frac{ET.SC}{US}\right)} \tag{4.35}$$

The minimum effective compressed download energy efficiency, *EE.CDW.min(s)*, is calculated as the uncompressed file size in megabytes, *US*, divided by the maximum energy to perform the compressed download, *ET.CDW.max(s)*. The final expressions in Equations (4.35) and (4.33) show the boundaries for the compressed download energy efficiencies as a function of the network parameters,

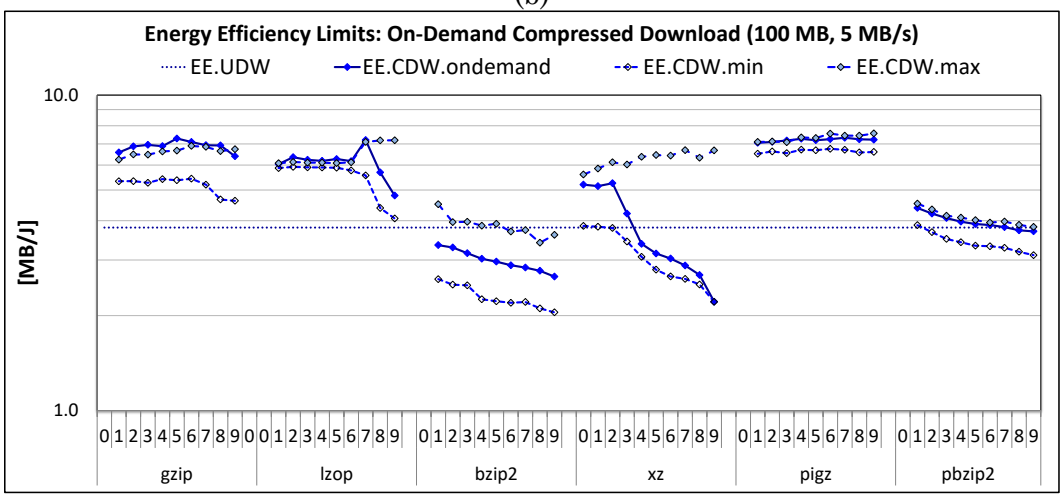
the file size, the compression ratio, the idle energy efficiency due to compression on the server, and the local decompression energy efficiency on the edge device.



(a)



(b)



(c)

Figure 4.18 Energy efficiency limits: compressed downloads with on-demand compression for 5 MB (a), 20 MB (b), 100 MB (c) files

4.2.2.3 Piping Model

Whereas we experimentally verified that we can estimate the minimum and maximum compressed transfer throughputs and energy efficiencies, the distance between these boundaries for a particular compression mode is often too wide, rendering them insufficient for estimation of effective throughputs or energy efficiencies. Ideally, we would like to be able to devise models for accurate estimation of effective upload and download throughputs and energy efficiencies.

Compressed uploads and downloads. The use of piping when transferring data file is beneficial as it increases the effective throughput and energy efficiency. It allows for overlapping local (de)compression tasks with the file transfer tasks on edge devices. In a case of compressed upload, a degree of this overlap depends on the ratio between the network upload throughput or energy efficiency, *Th.UP [EE.UP]*, and the local compression throughput or energy efficiency, *Th.C [EE.C]*. When the local compression throughput or energy efficiency exceeds by far the corresponding network upload throughput, the bottleneck is the network. When the local compression throughput or energy efficiency falls below the corresponding network throughput, the compressed upload is not beneficial. In a case of compressed downloads, when the compressed file is available on the server, a degree of overlap depends on the ratio between the network download throughput or energy efficiency, *Th.DW [EE.DW]*, and the local decompression parameter, *Th.D [EE.D]*.

$$k.th.c = \begin{cases} \frac{Th.UP}{Th.C}, & Th.C > Th.UP \\ 1, & Th.C < Th.UP \end{cases} \tag{4.36}$$

$$Th.CUP.pipe \approx \frac{CR \cdot Th.UP}{1 + CR \cdot Th.UP \cdot (\frac{k.th.c}{Th.C} + \frac{T.SC}{US})} \tag{4.37}$$

$$k.th.d = \begin{cases} \frac{Th.DW}{Th.D}, & Th.D > Th.DW \\ 1, & Th.D < Th.DW \end{cases} \quad (4.38)$$

$$Th.CDW.pipe \approx \frac{CR \cdot Th.DW}{1 + CR \cdot Th.DW \cdot (\frac{k.th.d}{Th.D} + \frac{T.SC}{US})} \quad (4.39)$$

To derive the piping model for upload throughput, the compression term from the lower throughput limit is restricted using a corrective factor, described in Equation (4.36). This factor lowers the impact of the local compression term when the local compression throughput exceeds the network connection upload throughput. The final model for the compressed upload throughput with the use of piping is expressed in Equation (4.38). To derive the piping model for download throughput, the decompression term from the lower throughput limit is restricted using a corrective factor, described in Equation (4.37). The final model for compressed download throughput is shown in Equation (4.39).

To derive the piping model for upload energy efficiency, the compression term from the lower energy efficiency limit is restricted using a corrective factor, described in Equation (4.40). To derive the piping model for the download energy efficiency, the decompression term from the lower energy efficiency limit is restricted using a corrective factor, as described in Equation (4.42). Effectively, the corrective factors restrict the energy component of the local (de)compression that includes the energy needed to run the platform, which is $ET.C - ET.C(0)$ for compression and $[ET.D - ET.D(0)]$ for decompression. The final models for the compressed upload and download energy efficiencies with piping are expressed in Equations (4.41) and (4.43), respectively.

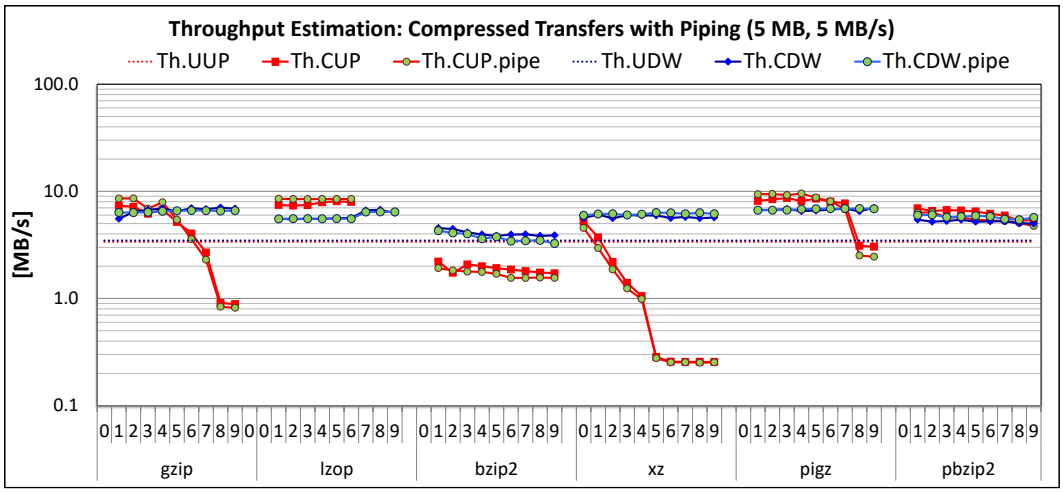
$$k.ee.c = \begin{cases} \frac{EE.UP}{EE.C}, EE.C > EE.UP \\ 1, EE.C < EE.UP \end{cases} \quad (4.40)$$

$$EE.CUP.pipe \approx \frac{CR \cdot EE.UP}{1 + CR \cdot EE.UP \cdot (\frac{k.ee.c}{EE.C} + \frac{1 - k.ee.c}{EE.C(0)} + \frac{ET.SC}{US})} \quad (4.41)$$

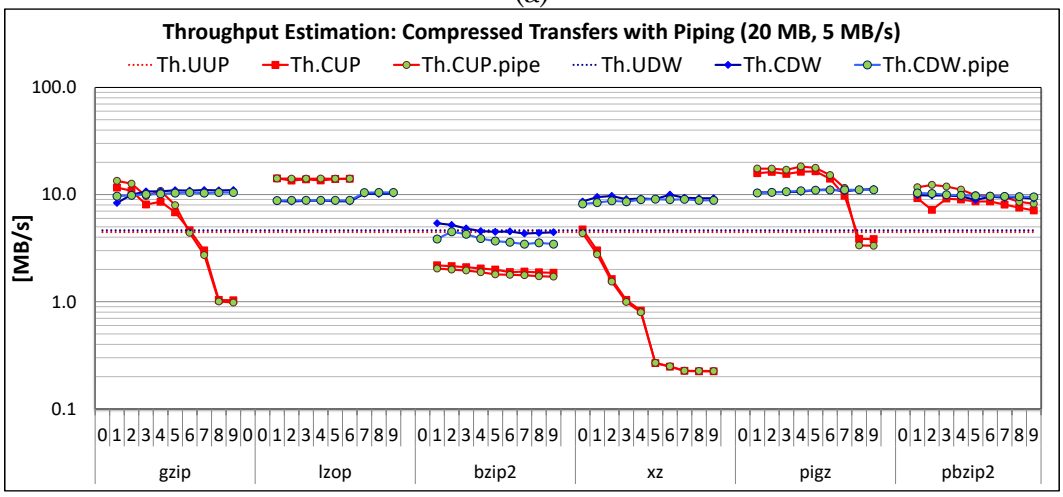
$$k.ee.d = \begin{cases} \frac{EE.DW}{EE.D}, EE.D > EE.DW \\ 1, EE.D < EE.DW \end{cases} \quad (4.42)$$

$$EE.CDW.pipe \approx \frac{CR \cdot EE.DW}{1 + CR \cdot EE.DW \cdot (\frac{k.ee.d}{EE.D} + \frac{1 - k.ee.d}{EE.D(0)} + \frac{ET.SC}{US})} \quad (4.43)$$

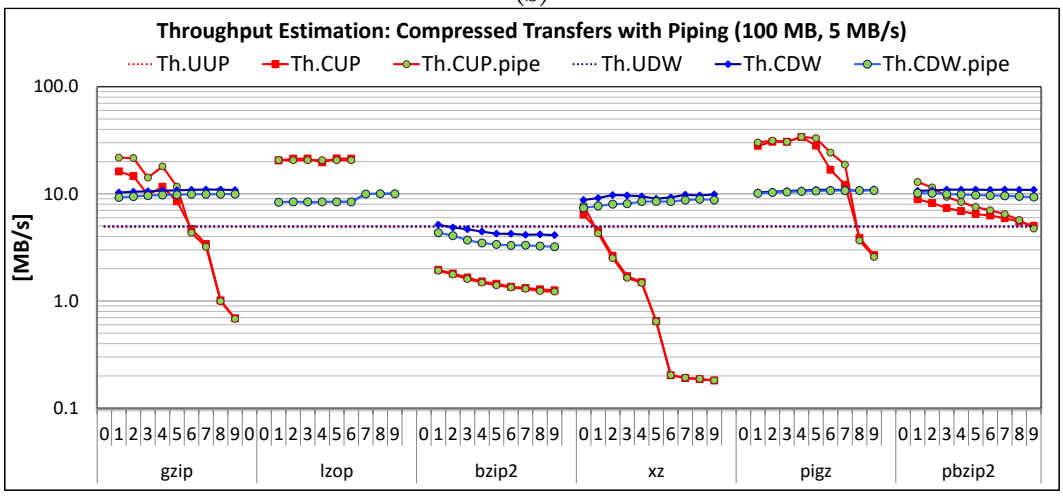
Figure 4.20 (a) shows the estimated compressed upload (green dots) and download (green circles) throughput and the measured compressed upload (red squares) and download (blue triangles) throughput for all considered compression modes. Figure 4.20 (b) shows the estimated compressed upload and download energy efficiency and the measured compressed upload and download energy efficiencies for all considered compression modes. The plots suggest a very high accuracy of the proposed models for all compression utilities and compression levels. This expression implies that if we know the parameters of the network connection (*Th.UP [Th.DW]* and *T.SC* or *EE.UP [EE.DW] and ET.SC*), and if for a given uncompressed file of size *US* we can predict the compression ratio, *CR*, and local (de)compression throughput or energy efficiency for a given (utility, level) pair (*Th.C [Th.D]* or *EE.C [EE.D]*) on a particular edge device, we can fairly accurately estimate the expected compressed upload or download throughput and energy efficiency.



(a)



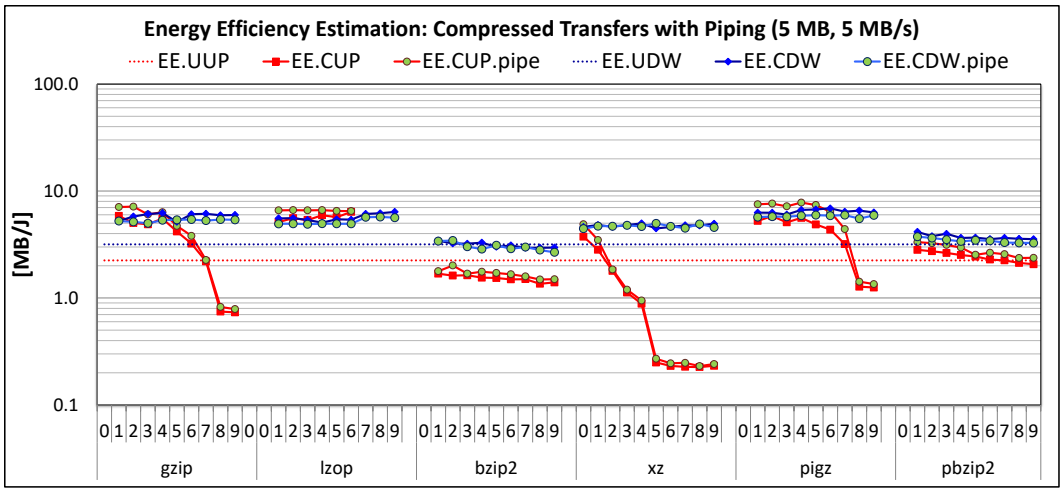
(b)



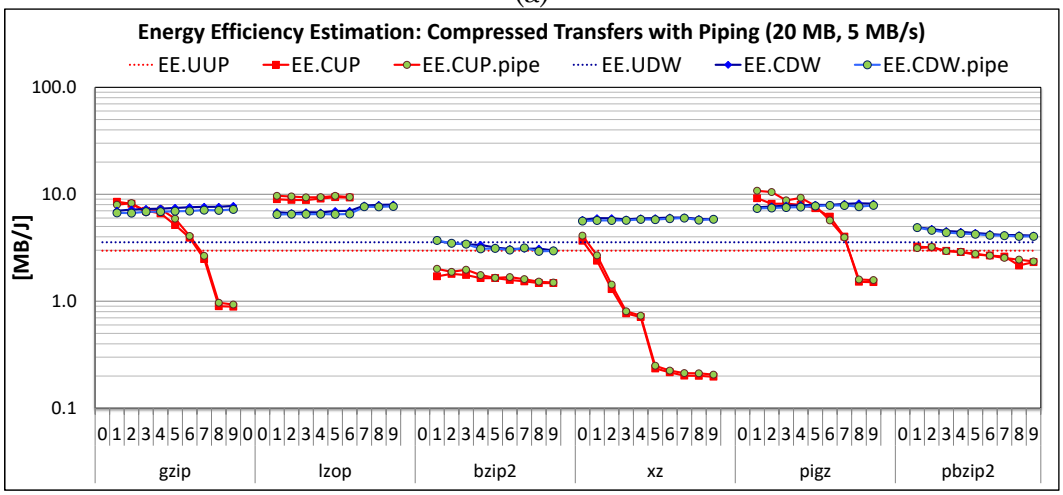
(c)

Figure 4.19 Compressed upload and download with piping:

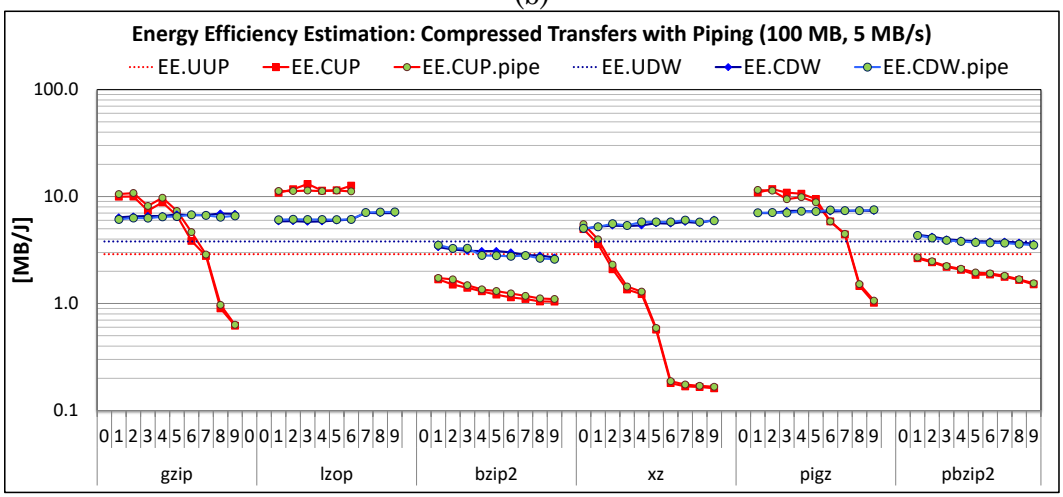
throughput estimation for 5 MB (a), 20 MB (b), and 100 MB (c) files



(a)



(b)



(c)

Figure 4.20 Compressed upload and download with piping:

energy efficiency estimation for 5 MB (a), 20 MB (b), and 100 MB (c) files

Compressed downloads (with on-demand compression). When the compressed file is not available on the server, the use of piping when downloading a file depends on the overlap between the compression tasks on the server, local decompression on the edge device, and file transfer tasks on the edge device. A degree of overlap depends on two ratios. The first overlap is between the network upload throughput (relative to the server), *Th.UP(s)*, and the server compression parameter, *Th.C(s)*. The second overlap is between the network download throughput or energy efficiency, *Th.DW [EE.DW]*, and the edge device's decompression parameter, *Th.D [EE.D]*.

$$k.th.c(s) = \begin{cases} \frac{Th.UP}{Th.C(s)}, & Th.C(s) > Th.UP \\ 1, & Th.C(s) < Th.UP \end{cases} \tag{4.44}$$

$$k.th.d = \begin{cases} \frac{Th.DW}{Th.D}, & Th.D > Th.DW \\ 1, & Th.D < Th.DW \end{cases} \tag{4.45}$$

$$Th.CDW.pipe \approx \frac{CR \cdot Th.DW}{1 + CR \cdot Th.DW \cdot (\frac{k.th.c(s)}{Th.C(s)} + \frac{k.th.d}{Th.D} + \frac{T.SC}{US})} \tag{4.46}$$

To derive the piping model for download throughput, the server's compression term and the edge device's decompression term from the lower throughput limit are restricted using corrective factors, described in Equations (4.44) and (4.45). The factor from Equation (4.44) lowers the impact of the server's local compression when the server's local compression throughput exceeds the network connection transfer throughput, *Th.C(s) >> Th.UP*. The factor from Equation (4.45) lowers the impact of the edge device's local decompression term when the local decompression throughput

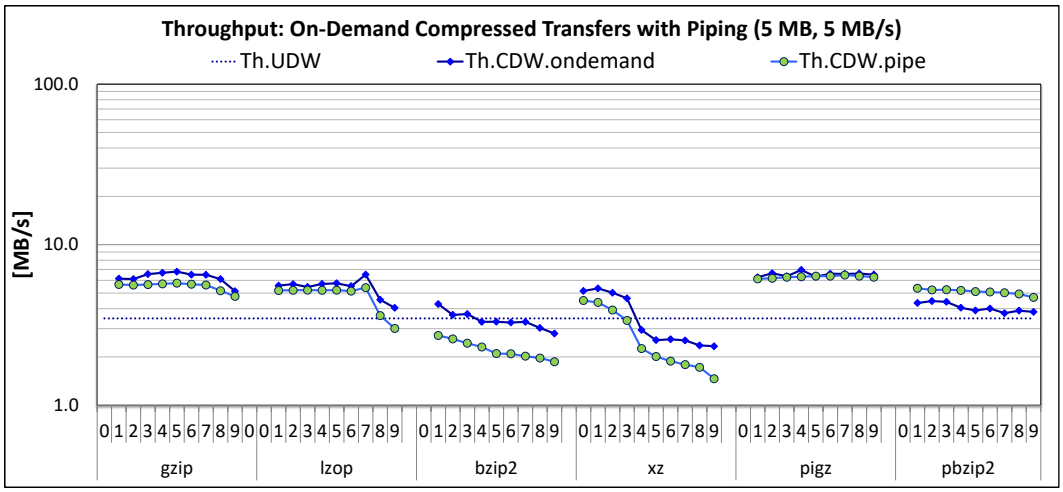
exceeds the network connection download throughput, $Th.D >> Th.DW$. The final model for compressed download throughput is shown in Equation (4.46).

To derive the piping model for download energy efficiency, the server's compression term and the edge device's decompression term from the lower energy efficiency limit are restricted using corrective factors, described in Equations (4.47) and (4.48). The factor from Equation (4.47) lowers the impact of idle energy efficiency from the server's local compression when the server's local compression energy efficiency exceeds the energy efficiency of the network connection transfer, $EE.C >> EE.UP$. The factor from Equation (4.48) effectively restricts the energy component of the edge device's local decompression that includes the energy needed to run the platform for decompression, $ET.D - ET.D(0)$, when the local decompression energy efficiency exceeds the network connection download energy efficiency, $EE.D >> EE.DW$. The final model for compressed download energy efficiency with piping is shown in Equation (4.49).

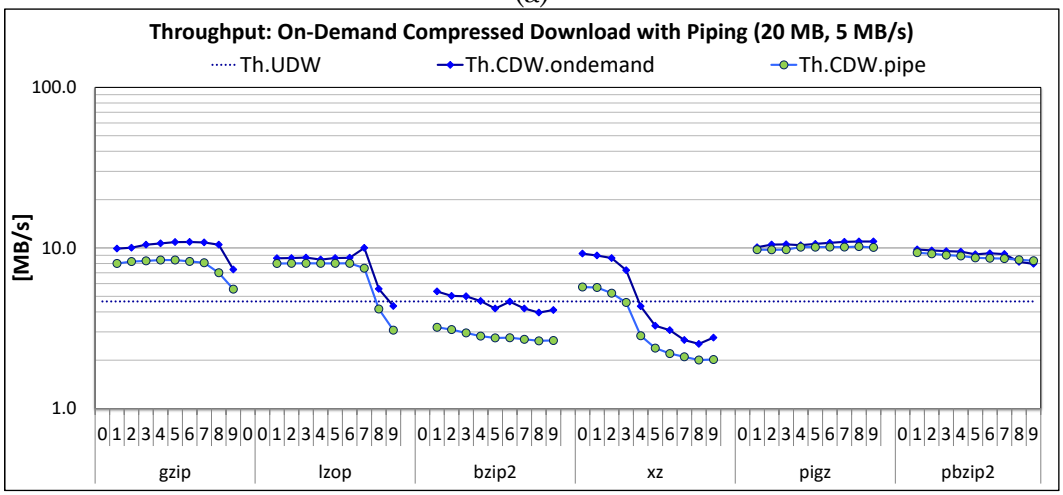
$$k.ee.c(s) = \begin{cases} \frac{EE.UP(s)}{EE.C(s)}, EE.C > EE.UP \\ 1, EE.C < EE.UP \end{cases} \tag{4.47}$$

$$k.ee.d = \begin{cases} \frac{EE.DW}{EE.D}, EE.D > EE.DW \\ 1, EE.D < EE.DW \end{cases} \tag{4.48}$$

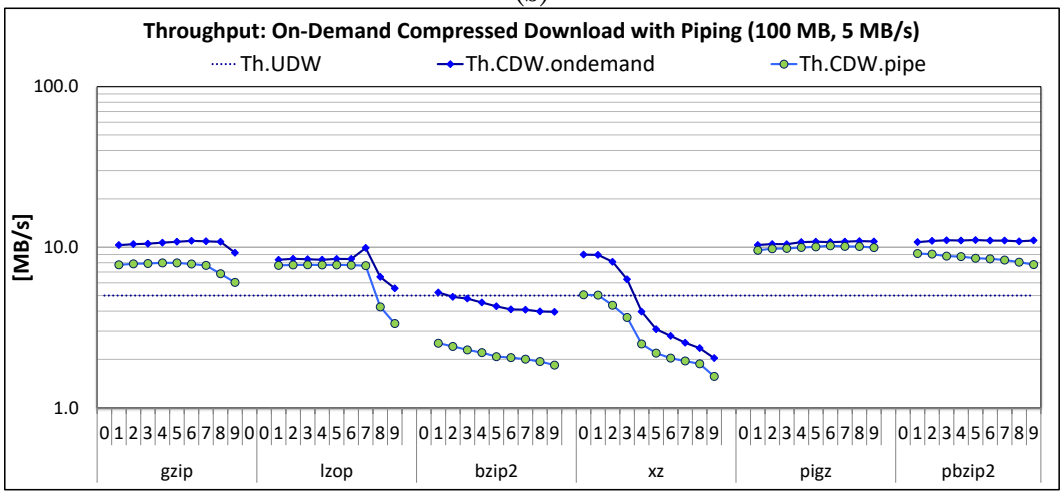
$$EE.CDW.pipe \approx \frac{CR \cdot EE.DW}{1 + CR \cdot EE.DW \cdot \left(\frac{k.ee.c(s)}{\frac{Th.C(s)}{V_{BS} \cdot I_{idle}}} + \frac{k.ee.d}{EE.D} + \frac{1 - k.ee.d}{EE.D(0)} + \frac{ET.SC}{US}\right)} \tag{4.49}$$



(a)

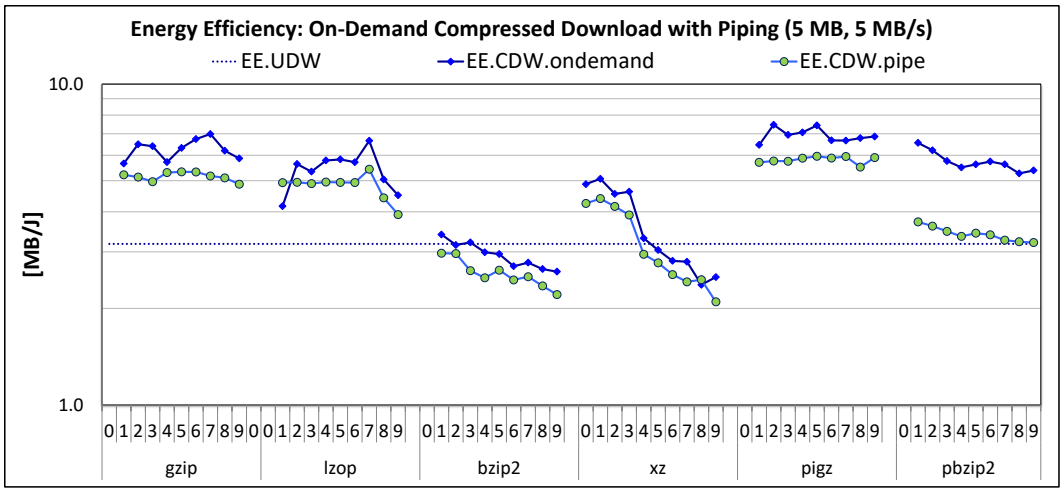


(b)

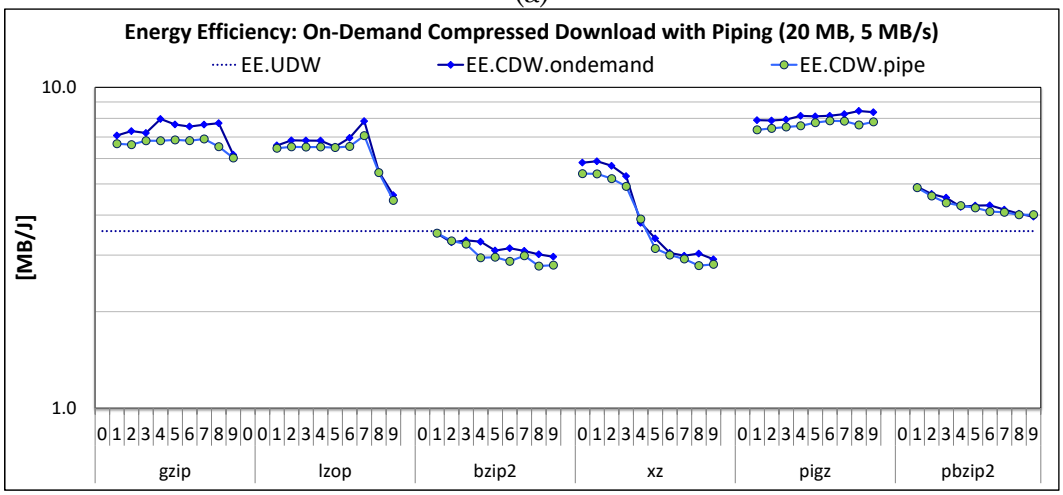


(c)

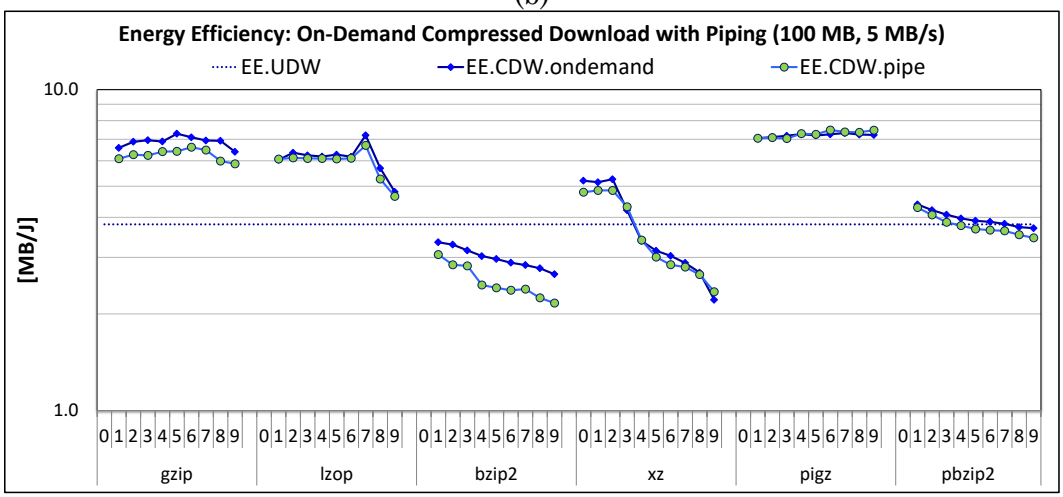
Figure 4.21 Compressed download with on-demand compression and piping: throughput for 5 MB (a), 20 MB (b), and 100 MB (c) files



(a)



(b)



(c)

Figure 4.22 Compressed download with on-demand compression and piping: energy efficiency for 5 MB (a), 20 MB (b), and 100 MB (c) files

4.2.2.4 Energy Estimation using Performance and Device Characteristics

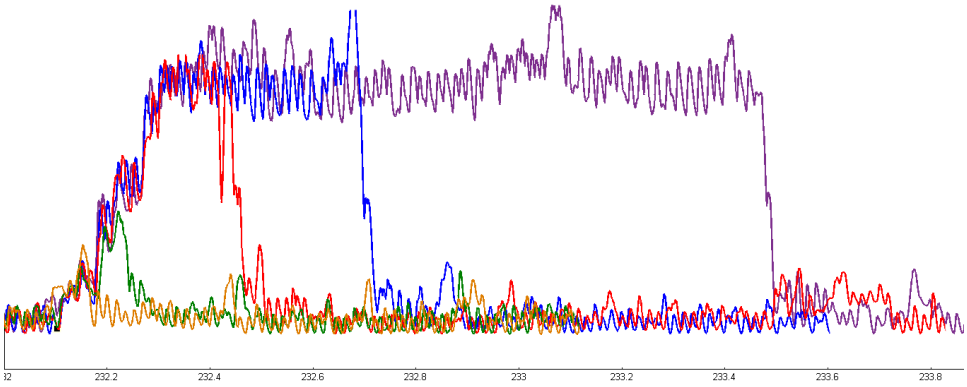
To select the most energy efficient mode of compressed transfer, without reliance on energy instrumentation, the energy models have to be estimated from the prediction data (*Th.C, US*) and set of device specific characteristics: (a) the device battery voltage, V_{BS}, (b) the idle current, I_{idle}, and (c) the active peak currents for each local (de)compression utility level pair, $I.C_{delta_{Max}}$ *[*$I.D_{delta_{Max}}$*]*). Similar to the estimation of network energy efficiency, the final expressions for estimated energy efficiency for local (de)compression, *EE.C [EE.D]* are shown in Equation (4.50) and (4.52) as a function of the effective local (de)compression throughput, the device battery voltage, and of idle and active peak currents. The final expressions for estimated energy overhead of local (de)compression, *EE.C(0) [EE.D(0)]*, are shown in (4.51) and (4.53). as a function of the effective local (de)compression throughput, the device battery voltage, and of active peak current. The difference from the estimation of network energy efficiency is the finite execution time of local (de)compression tasks. Thus, the active current may not always reach its maximum peak value due to the rise and fall of current. To accurately estimate energy efficiencies and energy efficiency overheads, the average active peak current has to be calculated, $I.C_{delta}$.

$$EE.C = \frac{US}{V_{BS} \cdot T.C \cdot (I_{idle} + I.C_{delta})} = \frac{Th.C}{V_{BS} \cdot (I_{idle} + I.C_{delta})} \tag{4.50}$$

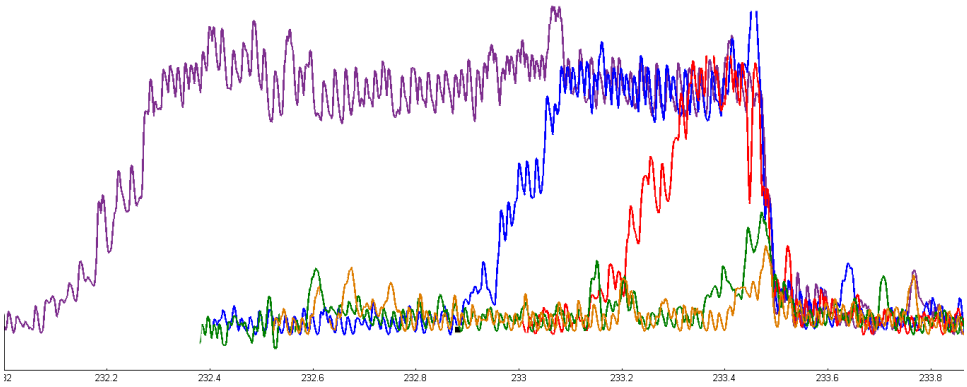
$$EE.C(0) = \frac{US}{V_{BS} \cdot T.C \cdot (I.C_{delta})} = \frac{Th.C}{V_{BS} \cdot (I.C_{delta})} \tag{4.51}$$

$$EE.D = \frac{US}{V_{BS} \cdot T.D \cdot (I_{idle} + I.D_{delta})} = \frac{Th.D}{V_{BS} \cdot (I_{idle} + I.D_{delta})} \tag{4.52}$$

$$EE.D(0) = \frac{US}{V_{BS} \cdot T.D \cdot (I.D_{delta})} = \frac{Th.D}{V_{BS} \cdot (I.D_{delta})} \tag{4.53}$$



(a)



(b)

Figure 4.23 Current waveforms for performing local compression using *pigz* -6 on set of varying file sizes – alignment to the starting (a) and ending (b) timestamp

To estimate average active peak current, $I.C_{delta}$, a set of measurement-based experiments have been conducted to record current waveforms for local (de)compression of files ranging in sizes from 0.1 MB to 45.15 MB. Figure 4.23 (a) and (b) show the recorded current waveforms for local *pigz* -6 compressions, with each waveform is color coded for the selected file size - e.g., orange line represents 0.1 MB file and purple line represents 45.16 MB file. To visually compare current

waveforms, Figure 4.23 (a) places current waveforms at the starting timestamp, while Figure 4.23 (b) places current waveforms at the ending timestamp. In both cases, we can see that all files have a similar rise and fall of current. Larger files achieve the maximum active peak current (red, blue and purple lines) and stay there for the duration of execution until a sudden drop to idle level. Extended measurement-based study has shown that the maximum active peak currents stay consistent for each (utility, level) pair and do not change with the changes in the input file type.

To calculate the energy overhead for local (de)compression, from which we can derive average active peak current, we have to either calculate an area of a trapezoid or an area of a triangle. To calculate energy overhead when the active peak current of executed task equals to its maximum active peak current, the equation is set to $A = \frac{1}{2} H_b \cdot b_1 \left(2 - \frac{H_b(\cot(\alpha)+\cot(\beta))}{b_1}\right)$ – with height, H_b, being the active peak current for the selected (utility, level) pair times the device battery voltage, V_{BS}, base, b_1, being the execution time, *T.C [T.D]*, and angles, α and β, corresponding to the inner angles of the rise and fall of current. To calculate energy overhead when the maximum active peak current is not reached by the executed task, the equation is set to $A = \frac{1}{2} H_b \cdot b_1$ – with height, H_b, being the active peak current times the device battery voltage, V_{BS}, and the base, b_1, being the execution time, *T.C [T.D]*).

The final expressions for averaged active peak current, $I.C_{delta}$ *[*$I.D_{delta}$*]*, are shown in Equations (4.54) and (4.55), respectively. The lower limit in Equation (4.54), $I.C_{delta_A}$, is calculated as a function of execution time, $T.C$, and the summation of cotangents of angle α and β, $\cot(\alpha)$ and $\cot(\beta)$, when it is less than the maximum active peak current. The upper limit Equation (4.54) is calculated as a function of execution time, $T.C$, the maximum active peak current, $I.C_{delta_{Max}}$, and the sum-

mation of cotangents of angle α and β, $\cot(\alpha)$ and $\cot(\beta)$, when the $I.C_{delta_A}$ is equal to or greater than the maximum active peak current, $I.C_{delta_{Max}}$. The lower and upper limits are express similarly for local decompression in Equation (4.55).

$$I.C_{delta} = \frac{1}{2}\begin{cases} I.C_{delta_A} = \frac{T.C}{\cot(\alpha)+\cot(\beta)}, I.C_{delta_A} < I.C_{delta_{Max}} \\ I.C_{delta_{Max}}(2 - \frac{I.C_{delta_{Max}}}{T.C}(\cot(\alpha) + \cot(\beta))), I.C_{delta_A} \geq I.C_{delta_{Max}} \end{cases} \quad (4.54)$$

$$I.D_{delta} = \frac{1}{2}\begin{cases} I.D_{delta_A} = \frac{T.D}{\cot(\alpha)+\cot(\beta)}, I.D_{delta_A} < I.D_{delta_{Max}} \\ I.D_{delta_{Max}}(2 - \frac{I.D_{delta_{Max}}}{T.D}(\cot(\alpha) + \cot(\beta))), I.D_{delta_A} \geq I.D_{delta_{Max}} \end{cases} \quad (4.55)$$

With the estimated average active peak current, the total energy and the energy overhead of local (de)compression tasks can be calculated. The energy efficiency models for compressed transfers are then used with estimated local energy efficiency, *EE.C [EE.D]*, estimated local energy efficiency overhead, *EE.C(0) [EE.D(0)]*, and the estimated energy network parameters.

4.2.2.5 Framework Utilization

During the framework utilization of models for compressed file transfers, besides the network parameters which are derived in Section 4.2.1.3, the only remaining unknowns in the models are the compression ratio, CR, the local (de)compression throughput, $Th.C$ [$Th.D$] for compression (utility, level) pair, the set of device specific characteristics (voltage, V_{bs}, idle current, I_{idle}), and the maximum active peak currents for local compression (utility, level) pairs and network transfers. Those unknown values will be estimated through a table look up quires, which will be discussed in the next two sections.

4.2.3 Collecting Local Compression and Decompression Prediction Data

The analytical models for estimating throughput and energy efficiency of compressed uploads and downloads require estimation of the compression ratio, CR, and local (de)compression throughputs, $Th.C$ *[Th.D]* through use of prediction tables containing historical logs of local (de)compression tasks performed on a number of edge devices and server platforms. The prediction tables with historical logs are generated through the measurement-based experiments to determine compression ratios, and local throughputs for local (de)compression of all data files from the selected datasets for mobile and workstation platforms introduced in Section 3.3. Historical logs from server platforms are generated to address models for on-demand compressed downloads, with the server-side compression. For framework utilization, we consider two edge devices, the OnePlus One smartphone and the Dell PowerEdge T110 II workstation, as well as local server, Dell PowerEdge T620, and the cloud instances in North Virginia and Tokyo. Each log entry keeps the following parameters: the compression utility and compression level used during (de)compression, the uncompressed file size (US), the compression ratio (CR), and the (de)compression throughputs ($Th.C$ [$Th.D$]).

Figure 4.24 (a) illustrates the relationship between the uncompressed file size, US, and the compression ratio, CR, and Figure 4.24 (b) shows the relationship between the uncompressed file size, US, and local compression throughput, $Th.C$, when compressing various input files using the *gzip* with -1 utility on the OnePlus One smartphone. The results indicate that both the compression ratio, CR, and the local compression throughput , $Th.C$, can be predicted due to a linear or a curved clustering of the collected data points for each distinct file type and compression

(utility, level) pair. The input file size and file type are used as input parameters for the prediction table lookup queries. The estimated *CR, Th.C, and Th.D* values from the table look up are then used in the previously described analytical models to derive actual throughputs or to estimate energy efficiencies for compressed uploads or downloads with all available compression (utility, level) pairs.

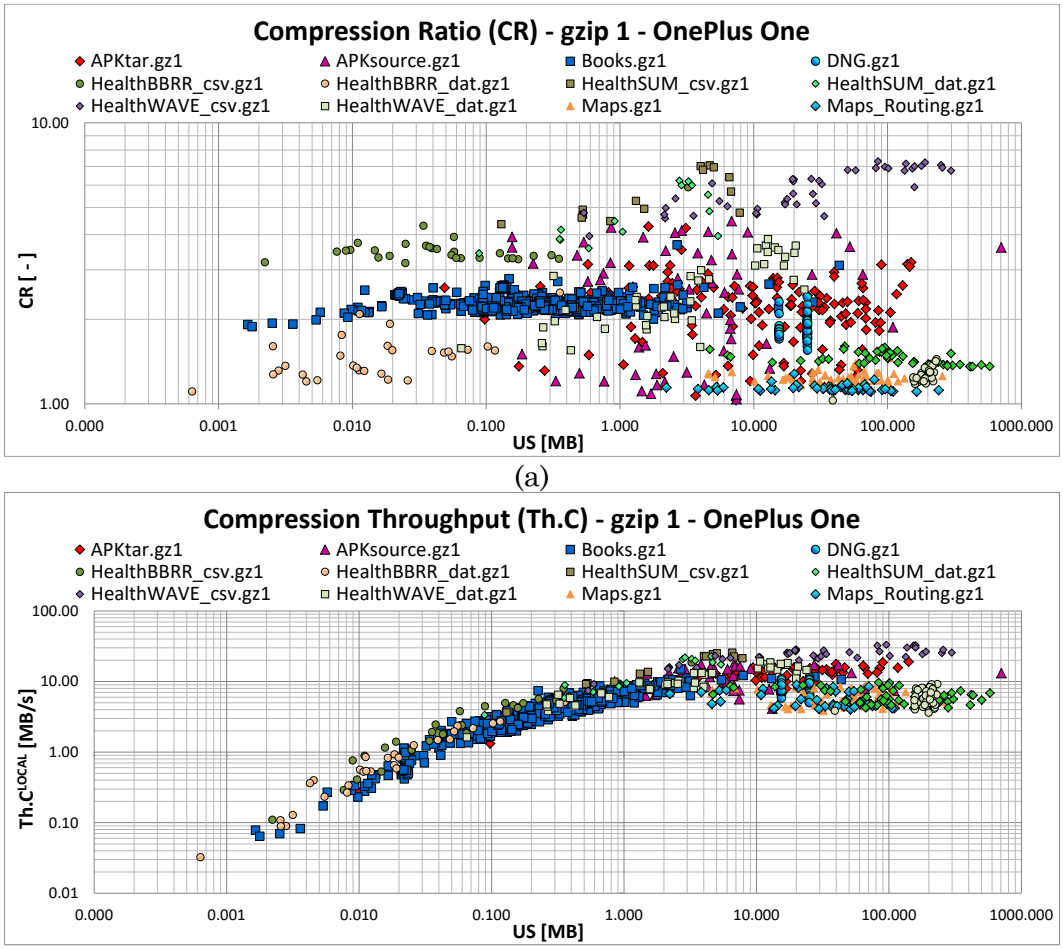


Figure 4.24 CR and Th.C to change in US – compression (OnePlus One)

4.2.4 Prediction Data Tables

The edge device and cloud agents maintain a set of data tables to perform a set of functions for selection of optimal data transfers. Sub-section 4.2.4.1 describes data table structures used by the agents on edge device and the cloud. Sub-section 4.2.4.2 explains life cycle of data tables during framework utilization. Sub-section 4.2.4.3 covers several database optimization techniques used for speeding up query executions.

4.2.4.1 Table Structure

The data tables maintained by the edge device agent are *historyComp*, *historyDecomp*, *compUtil*, and *exclusionType* tables. The *historyComp* table contains historical logs used for predictions of local compressions and includes the following fields: *US* and *Type* to describe the uncompressed file size and type, *C_Util* and *C_Level* to describe the corresponding compression (utility, level) pair (e.g., "*gzip*", "1"), and *Th_C* and *CR* to describe the local compression throughput and compression ratio. The *historyDecomp* table similarly maintains history logs used for predictions of local decompressions and includes the following fields: *US* and *Type* to describe the uncompressed file size and type, *D_Util* and *D_Level* to describe the corresponding decompression (utility, level) pair, and *Th_D* and *CR* to describe the local decompression throughput and compression ratio. The *historyDecomp* table is update with each successful compressed download, and will be used for periodic update of the server's decompression records in a corresponding table. The *compUtil* table maintains a list of all compression (utility, level) pairs available on the device, as well as corresponding maximum active peak currents, *I_delta_Max*, and rise and fall angles of the current waveforms, *Angle_A*and *Angle_B*. The *compUtilExt* table

maintains files extensions of all compression utilities (e.g., *.gz*, *.bz2*, *.lzo*, *.xz*). Finally, the *exclusionType* table maintains a list of uncompressible file types which are ignored by the framework and instead transferred uncompressed.

The server's agent maintains a similar set of tables to the tables on the edge device. The distinction is that the server's agent maintains history logs for each serviceable edge devices, as well as for itself, and requires additional tables to perform long-term file storage and initialization of edge devices. The complete table structure including *historyComp, historyDecomp, historyCompServ, exclutionType, deviceType, and fileStroage* tables. The *historyComp* and historyDecomp tables contain historical logs used for predictions of local compressions and decompressions on the serviceable edge devices, and contain the same fields as on the corresponding edge device tables, but with an addition of *Device_ID* field, which stores a unique identifier of each serviceable edge device. The *historyCompServ* table contains historical logs used for predictions of local compressions on the server used with the on-demand download model. The table contains the same fields as on the corresponding edge device table. The *compUtil* table maintains a list of all compression (utility, level) pairs available on the serviceable edge devices, as well as corresponding maximum active peak currents, *I_delta_Max*, and rise and fall angles of the current waveforms, *Angle_A* and *Angle_B*. To track different edge devices, the *compUtil* tables include an additional *Device_ID* field. The *compUtilExt* table maintains files extensions of all compression utilities (e.g., *.gz*, *.bz2*, *.lzo*, *.xz*). The *exclusionType* table maintains a list of uncompressible file types which are ignored by the framework and instead transferred uncompressed. To maintain information about serviceable edge devices by the server, the *deviceType* table includes a list of all supported devices (joined

with other tables via *Device_ID* field). The table also includes additional fields such as device type and model. Finally, the *fileStorage* table maintains physical locations of available files on the server. The table include the following fields: *fileName* and *Type* to specify file names and types, *VersionNumber* to specify versions of the files that are currently available on the server (including uncompressed and compressed versions of the file), and *LocalPath* to specific file location on the server. The *fileStorage* table will be updated on each upload to the server and during the set of operations performed by the server such as re-compression and optimization of maintained storage.

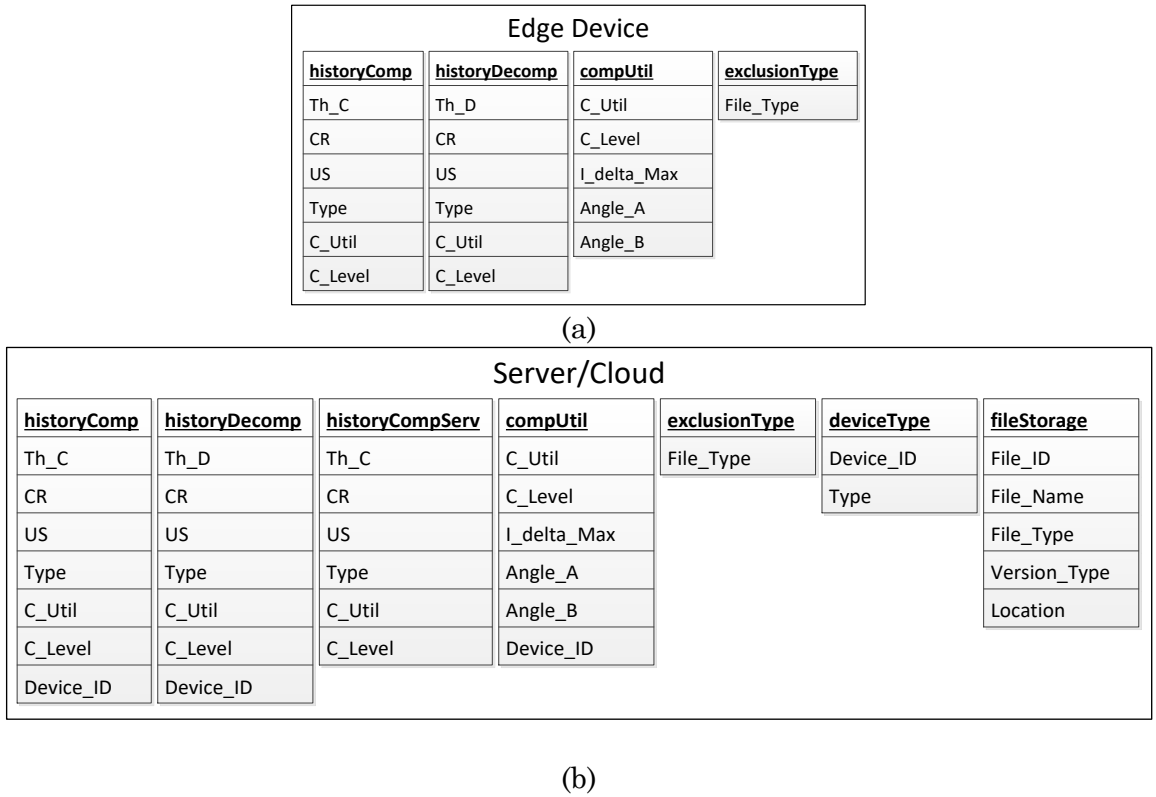


Figure 4.25 Prediction tables for edge device (a) and server/cloud (b)

The initial contents of *historyComp* and *historyDecomp* tables are generated from number of edge devices under-test with a collection of test input data for generation of initial prediction data for the edge devices. The information about used edge devices under-test will populate the *deviceType* table. Similarly, the initial content of *historyCompServ* table is generated from local server with the same collection of test input data for generation of server's prediction data for compression. The test input data would depend on type of application where framework will be utilized and implemented. The additional task of the server, besides satisfying download request and processing file storage on upload, includes the initial provisioning of *historyComp* and *historyDecomp* tables on the client-side agents running the edge devices with the same *Device_ID* or type. For this, the server's *historyComp* and *historyDecomp* tables will be used for initial provisioning, filtered to the specific *deviceID* or type of the provisioning device.

4.2.4.2 Database Life Cycles

To be able to provide optimal decision making for particular device, user, or application, data tables are continually updated and refined on both the client and the server. Figure 4.26 shows the life cycle of the framework's data tables and the ways how the data tables are updated and refined with each new upload, download, or other processes executed on the client or server-side of the framework.

(S1) On first initialization of the framework on the client, the server populates the client's data tables with default data. (C1) When the upload is performed from the client, *historyComp* table is updated with a new entry containing execution time of local compression ($Th.C$), compression ratio (CR), and uncompressed size of

the uploaded file (US) together with compression utility and a level that was selected. (S2) When a download is requested by the client, the server selects an optimally compressed file by performing a query on *historyDecomp* table and sends it to the requester. (C2) The client follows by receiving the files and updating its local *historyDecomp* table with estimated throughput of local decompression ($Th.D$), compression ratio (CR), and the size of the uncompressed file (US), together with the compression utility and compression level that was used on the server.

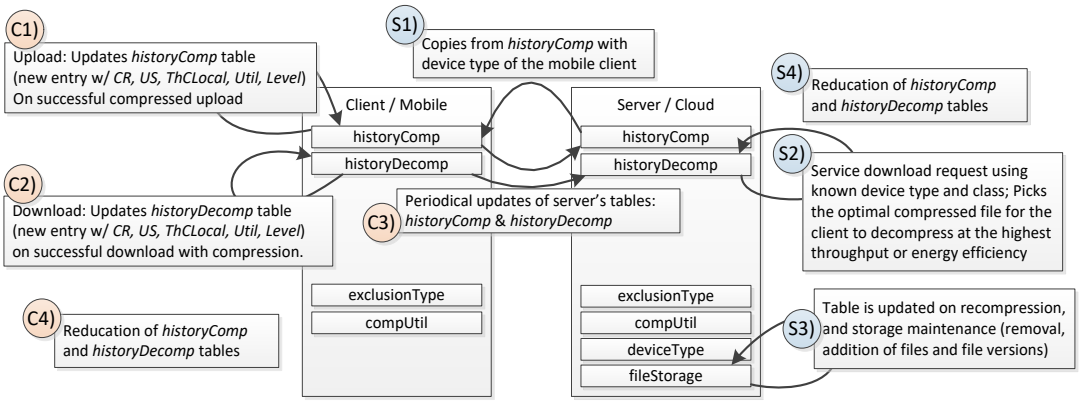


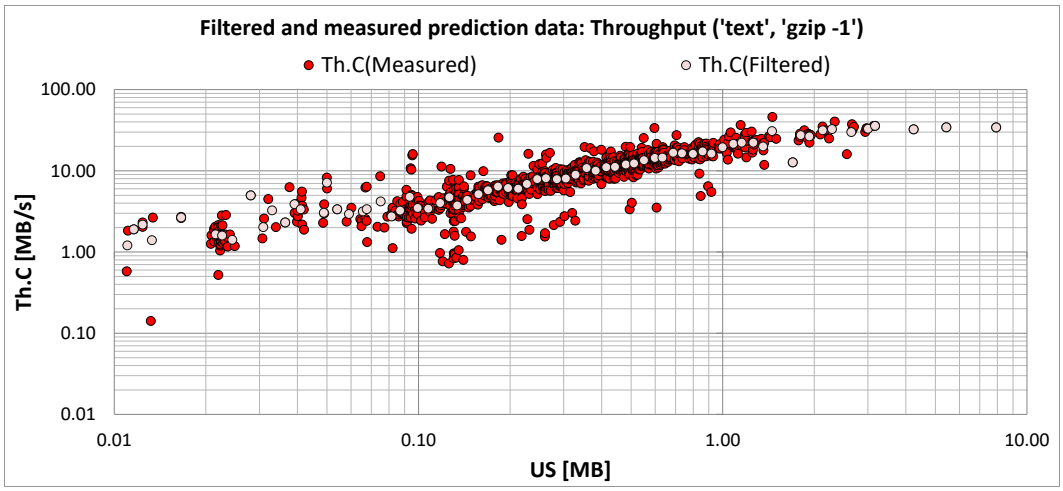
Figure 4.26 Framwork life cycle (C – client, S – server)

(C3) When a client is inactive, it will periodically update server's *historyComp* and *historyDecomp* tables with new values, which are generated from n number of previously completed uploads and downloads. Additionally, (C4 and S4), both the client and the server will perform periodic optimizations of their tables (specifically for *historyComp* and *historyDecomp* tables) by performing reduction and averaging of their values. This optimization will try to keep a number of records small for fast queries while striving to maintain granularity specifically for upload and download of small-

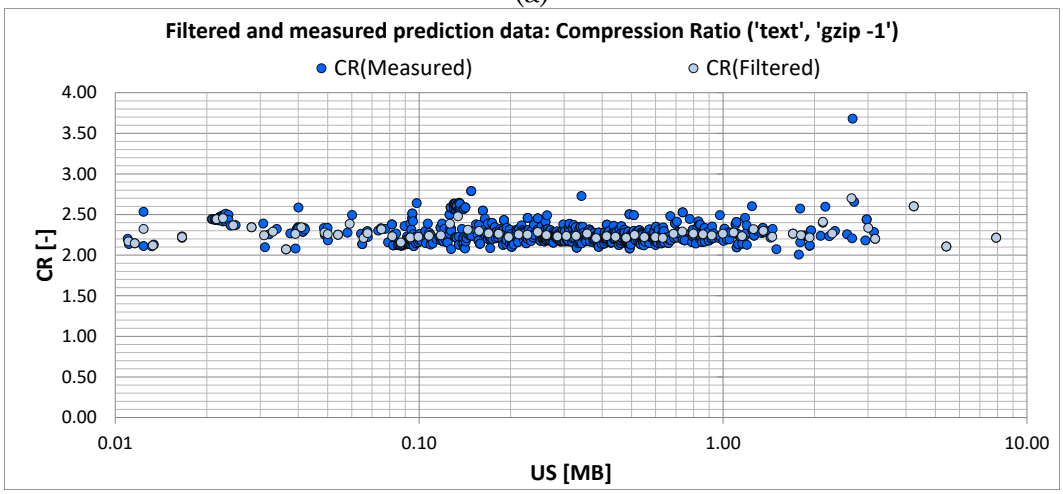
er files (under 1 MB in size). Finally, the server will update *fileStorage* table during file maintenance and recompression (S3).

4.2.4.3 Database Optimization

To speed up the execution of SQL queries, the number of entries in the tables is periodically reduced on both the mobile device and the server. The reduction is done over a logarithmically spaced array of file sizes, with an aim of providing high granularity for smaller files and lower granularity for larger files. Figure 4.27 shows the reduced and the original entries collected on OnePlus One smartphone for a 'text' dataset and *gzip* -1 compression. We show the relationship between the uncompressed file size, *US*, and the compression ratio, *CR*, and to the local compression throughput, *Th.C*. A continuous reduction will improve estimation of the requested parameters while allowing for fast queries. The agents also utilize a database index, *I1,* to speedup SQL queries by indexing all filtered fields during queries, including *US*, *Type*, *C_Util*, *C_Level*, and *Device_ID*.



(a)



(b)

Figure 4.27 Prediction tables for mobile device (a) and server/cloud (b)

CHAPTER 5

FRAMEWORK IMPLEMENTATION

To perform an evaluation of the framework during uploads and downloads, the framework's edge device and cloud agents have been implemented in C++ using SQLite and MySQL databases for table structures. For mobile devices, the client-side agent is compiled with Android NDK toolset to allow its native execution on Android, and SQLite database is selected due to its portability, speed, and default availability on Android. For workstations, the client-side agent can be implemented using any SQL engine as there are no constraints in development tools and environments. However, SQLite has been used to have a consistent implementation across all edge devices. The C++ implementation of the cloud agent is compiled with *gcc* and MySQL to allow concurrent request from the numerous devices. The database prediction tables are created for both the edge device and the cloud portions of the framework, containing the data tables described in the previous section (Section 4.2.4.1). The following subsections describe details of the framework implementation for uploads (Section 5.1) and downloads (Section 5.2). Section 5.3 describes methods for server's long-term storage of the servisable files, including re-compression of files done on the server.

5.1 Upload Agents

The framework's optimization of uploads involves two agents, the agent residing on the edge device (client), and the agent residing on the server or the cloud. During uploads, the agent on the edge device performs all the decision making to select the most effective mode of file upload. The server's agent is responsible for storage management, receiving and processing incoming uploads, and finally, performing re-compression tasks to make new compressed versions of files available to clients. The primary examples of file uploads performed from the edge devices can be file distribution, mHealth data upload, or general file uploads.

To perform optimized upload, the first step for the client is to determine the current network parameters, including network throughput, *Th.UP*, and connection setup time, *T.SC*. To avoid imposing an additional latency to the current file upload, these parameters are typically acquired prior to upload – e.g., an agent can periodically probe network conditions using the method discussed in Section 4.2.1.3. Once the throughput-based network parameters are calculated, energy efficiency network parameters are estimated using estimation models from Section 4.2.1.1.

Figure 5.1 (line 2) shows command arguments for invoking the framework agent for the throughput optimized upload of a text file over a WLAN interface with the network throughput set to 5 MB/s and the connection time of 0.362 seconds. Figure 5.1 (line 4) shows command arguments for invoking the framework agent for energy efficiency optimized upload of a text file over a WLAN interface with the network throughput set to 0.5 MB/s and the connection time of 0.362 seconds. In both cases, the network parameters are determined in advance by the network probing. The file information (file size and type) is retrieved from the file system of the edge

device when invoking the upload agent. These inputs are then used in an SQL query to estimate compression ratios, *CR*, and local compression throughputs, *Th.C*, for all available compression utilities and levels on the mobile device. If the framework is using throughput mode, the framework estimates the throughput of compressed uploads using the analytical model for *Th.CUP*. If the framework is using energy-efficient mode, the framework estimates energy efficiency of compressed uploads using the analytical model for estimated energy efficiency, *EE.CUP*.

```
1. // Throughput based upload @ 5 MB/s network throughput
2. frameworkRUN –upload –TH 'fileName.txt' 'text'0.362 5.11
3. // Energy efficiency based upload @ 0.5 MB/s network throughput
4. frameworkRUN -upload –EE 'fileName.txt' 'text'0.362 0.5113
```

Figure 5.1 Upload of a text file from the client over 5 MB/s and 0.5 MB/s networks, with throughput (TH) and energy efficiency (EE) modes

Figure 5.2 shows a SQL query which is executed inside the framework agent on throughput optimized upload of a text file from the command argument in Figure 5.1 (line 2). The results of the SQL query (Table 5.1) are sorted in the descending order, with a limit applied to the first row, which has the highest compressed upload throughput, *Th.CUP*. If the highest compressed upload throughput exceeds the uncompressed upload throughput, *Th.UUP*, the selected utility and level are used during compressed upload; otherwise, the uncompressed file is uploaded.

For the throughput mode, the SQL query execution with a 30,000 entry table is 27.4 milliseconds on the OnePlus smartphone, and between 2 and 3.5 milliseconds on the workstation.

```
1. // inUS=1.77; inType='text'; inThUP=5.0; inTSC=0.362;
2. SELECT T.C_Util,T.C_Level,Ext,CR,Th_C FROM (
3. SELECT C_Util,C_Level,(SELECT CR FROM (SELECT CR, diff FROM (
4.    SELECT CR, abs(US - inUS) AS diff FROM historyComp AS T2
5.    WHERE Type = inType and T2.C_Util = Tools.C_Util and
6.    T2.C_Level = Tools.C_Level and US <= inUS
7.    ORDER BY US DESC LIMIT 1)UNION ALL
8.  SELECT CR, diff FROM (
9.    SELECT CR, abs(US - inUS) AS diff FROM historyComp AS T2
10.   WHERE US= inType and T2.C_Util = Tools.C_Util and
11.   T2.C_Level = Tools.C_Level and US > inUS ORDER BY US DESC LIMIT 1)
12.   ORDER BY diff LIMIT 1)
13.) AS CR,(SELECT Th_C FROM (SELECT Th_C, diff FROM (
14.   SELECT Th_C, abs(US - inUS) AS diff FROM historyComp AS T3
15.   WHERE Type = inType and T3.C_Util = Tools.C_Util and
16.   T3.C_level = Tools.C_Level and US <= inUS
17.   ORDER BY US DESC LIMIT 1) UNION ALL
18. SELECT Th_C, diff FROM (
19.   SELECT Th_C, abs(US - inUS) AS diff FROM historyComp AS T3
20.   WHERE Type = inType and T3.C_Util = Tools.C_Util and
21.   T3.C_Level = Tools.C_Level and US > inUS ORDER BY US DESC LIMIT 1)
22.   ORDER BY diff LIMIT 1)) AS Th_C FROM compUtil AS Tools) AS T
23.LEFT JOIN compUtilExt ON (compUtilExt.C_Util = T.C_Util)
24.ORDER BY (1/(inTSC/US+inThUP/(Th_C*Th_C)+1/(CR*inThUP)) DESC LIMIT 1
```

Figure 5.2 SQL query for uploading 1.75 MB file with optimized throughput

Table 5.1 SQL query output sorted for throughput optimized upload @ 5 MB/s

i	Utility	Level	*Th.C*	*Th.CUP*
1	lzop	1	18.513	**3.021**
2	lzop	6	15.437	**2.955**
3	pigz	1	9.340	**2.877**
4	gzip	1	8.636	**2.799**
5	pigz	6	5.128	**2.150**
6	pigz	9	4.641	**1.975**
7	xz	0	3.449	**1.431**
8	gzip	6	3.010	**1.215**
9	bzip2	1	2.442	**0.913**
10	xz	1	2.159	**0.749**

Figure 5.3 shows a SQL query which is executed inside the framework agent on energy efficiency optimized upload of a text file from the command argument in Figure 5.1 (line 4). The results of the SQL query (Table 5.2) are sorted in the descending order, with a limit applied to the first row, which has the highest compressed upload energy efficiency, *EE.CUP*. If the highest compressed upload energy efficiency exceeds the uncompressed upload energy efficiency, *EE.UUP*, the selected utility and level are used during compressed upload; otherwise, the uncompressed file is uploaded. The SQL query execution with an 30,000 entry table on the OnePlus One for energy efficiency optimized transfers is 63.24 ms. The extra overhead comes from the additional SQL CASE statement used for calculation of EE_C. Execution of the same query on the workstation is between 3.0 ms and 6.5 ms.

```
1. // inUS=1.77; inType='text'; inThUP=0.50;  inTSC=0.362;
2. //  Vbs=4.10; I_idle=0.067;     estimated: inEEUP=0.96
3. SELECT T.C_Util,T.C_Level,Ext,CR,Th_C,
4. CASE WHEN (inUS/Th_C)/0.577 < T.Delta_I THEN
5. Th_C/(Vbs*(I_idle + 0.5*(inUS/Th_C)/0.577))
6. ELSE
7. Th_C/(Vbs*(I_idle + 0.5*(T.Delta_I*(2-T.Delta_I/(inUS/Th_C)*0.577 ))))
8. END AS EE_C FROM (SELECT C_Util,C_Level,Delta_I,
9. (SELECT CR FROM (SELECT CR, diff FROM (
10.    SELECT CR, abs(US - inUS) AS diff FROM historyComp AS T2
11.    WHERE Type = inType and T2.C_Util = Tools.C_Util and
12.    T2.C_Level = Tools.C_Level and US <= inUS
13.    ORDER BY US DESC LIMIT 1) UNION ALL
14.  SELECT CR, diff FROM (
15.    SELECT CR, abs(US - inUS) AS diff FROM historyComp AS T2
16.    WHERE Type = inType and T2.C_Util = Tools.C_Util and
17.    T2.C_Level = Tools.C_Level and US > inUS
18.    ORDER BY US DESC LIMIT 1) ORDER BY diff LIMIT 1)) AS CR,
19.(SELECT Th_C FROM (SELECT Th_C, diff FROM (
20.    SELECT Th_C, abs(US - inUS) AS diff FROM historyComp AS T3
21.    WHERE Type = inType and T3.C_Util= Tools.C_Util and
22.    T3.C_Level = Tools.C_Level and US <= inUS
23.    ORDER BY US DESC LIMIT 1) UNION ALL
24.  SELECT Th_C, diff FROM (
25.    SELECT Th_C, abs(US - inUS) AS diff FROM historyComp AS T3
26.    WHERE Type = inType and T3.C_Util = Tools.C_Util and
27.    T3.C_Level = Tools.C_Level and US > inUS ORDER BY US DESC LIMIT 1)
28.    ORDER BY diff LIMIT 1)) AS Th_C FROM compUtil AS Tools) AS T
29.LEFT JOIN compUtilExt ON (compUtilExt.C_Util = T.C_Util)
30. ORDER BY (1/(1/(CR*inEEUP)+1/EE_C+(inEEUP/EE_C-1)
    *(Vbs*I_idle)/Th_C+inTSC/inUS)) DESC LIMIT 1
```

Figure 5.3 SQL query for uploading 1.75 MB file with optimized energy efficiency

Table 5.2 SQL query output sorted for energy efficiency optimized upload @ 0.5 MB/s

i	Utility	Level	*EE.C*	*EE.CUP*
1	pigz	1	9.770	**1.610**
2	gzip	1	9.431	**1.599**
3	lzop	1	29.977	**1.463**
4	lzop	6	22.495	**1.429**
5	pigz	6	3.482	**1.352**
6	pigz	9	2.995	**1.271**
7	xz	0	3.136	**1.241**
8	gzip	6	2.865	**1.236**
9	bzip2	1	2.212	**1.154**
10	gzip	9	2.026	**1.029**

In both cases, if a compressed upload is selected, the compressed size, *CS*, and total execution time, *T.CUP*, are returned at the end of the upload process. The agent will then calculate compression ratio, *CR*, from the compressed file size, and estimate local compression throughput, $Th.C$, by using the model for compressed upload with piping shown in Equation (5.1). Finally, the uncompressed size, compression ratio, and local compression throughput together with corresponding compression utility and level are inserted as a new entry in the *historyComp* table. This will refine historical data table with each compressed upload transaction.

$$Th.C = \sqrt{\frac{Th.UP}{\frac{1}{Th.CUP} - \frac{T.SC}{US} - \frac{1}{CR \cdot Th.UP}}} \tag{5.1}$$

5.2 Download Agents

The framework's optimization of downloads involves two agents, the agent residing on the edge device (client), and the agent residing on the server or the cloud. During downloads, the agent on the server performs all the decision making to select the most effective mode of file download. The cloud agent can also perform on-demand compression, if the compressed versions of the requested files are not available in the *fileStorage* table. The agent on the edge device initiates the download request, and performs automatic decompression of the incoming file when needed. The primary examples of file downloads performed from the edge devices are downloads of applications and other executables from software repositories, downloads of processed mHealth data, and downloads of text and HTTP data.

To perform optimized download, the first step for the client is to determine the current network parameters, including network throughput, *Th.DW*, and the time to set up a network connection, *T.SC*. As in uploads, we assume that the client maintains the network parameters through periodic probing to avoid imposing an additional latency to the current file download. Similarly, both throughput and energy efficiency network parameters are estimated.

Figure 5.4 (line 2) shows command arguments for invoking the framework agent with throughput optimized download of a text file over a WLAN interface with the network throughput set to 5 MB/s and the connection time of 0.362 seconds. Figure 5.4 (line 4) shows command arguments for invoking the framework agent with energy efficiency optimized download of a text file over a WLAN interface with the network throughput set to 0.5 MB/s and the connection time of 0.362 seconds. In both cases, the network parameters are determined in advance by the network probing. The file information (file size and type) and device ID (e.g., A0001 for OnePlus One) are retrieved from the file system and operating system running on the edge device. Two additional inputs tell the framework agent to either perform download with optimized throughput (line 2) or download with optimized energy efficiency (line 4). When server receives the download request with client's information, it executes an SQLite query on *historyDecomp* table to decide which file (compressed or uncompressed) is the best in terms of performance or energy efficiency specifically for device type of the requesting client. If the download request favors the throughput optimized mode, the framework estimates the throughput of compressed downloads using the analytical model for *Th.CDW*. If the download request favors the energy efficiency optimized mode, the framework estimates energy efficiency of compressed downloads using the analytical model for estimated energy efficiency,

EE.CDW. When the compressed files of requested file download are not available the *fileStorage* table, the server agent will instead estimate the throughput or energy efficiency of compression downloads with on-demand compression.

Once the client agent starts receiving the file, it will decode the first few incoming bytes (magic number from Table 2.1) to decide which decompression, if any, should it use to complete the transaction. Figure 5.5 shows a part of the script used by the client for decoding incoming data. The framework can also be configured to use custom IDs for deciding which compression utility and compression level is needed for performing decompression.

```
1. // Throughput based download @ 5 MB/s network throughput
2. frameworkRUN –download –TH 'fileName.txt' 'text' 'mako' 0.362 5.11
3. // Energy efficiency based upload @ 0.5 MB/s network throughput
4. frameworkRUN -download –EE 'fileName.txt' 'text' 'mako' 0.362 0.5113
```

Figure 5.4 Download of text file from the client on 5 MB/s and 0.5 MB/s network throughput, with throughput (TH) and energy efficiency (EE) modes

```
1. !/bin/sh
2. trap "rm -f /tmp/$$; exit 1" 1 2 3 15
3. # grab the 1st 4 bytes off the input stream, store them in a file,
4. # convert to ascii, and store in variable:
5. HEADER=$(
6.      dd bs=1 count=2 2>/dev/null |
7.      tee /tmp/$$ |
8.      od -t x1 |
9.      sed '
10.         s/^00* //
11.         s/ //g
12.         q
13.     '
14.)
15.case "$HEADER" in
16.    1f8b)
17.       UNCOMPRESS='gzip -d -fc';;
18.    425a)
19.       UNCOMPRESS='bzip2 -d -fc';;
20.    894c)
21.       UNCOMPRESS='lzop -d -fc';;
22.    fd37)
23.       UNCOMPRESS='xz -d -fc';;
24.    *)
25.       UNCOMPRESS='cat';;
26.esac
27.#echo >&2 "$0: File header is '$HEADER' using '$UNCOMPRESS'
       on stream."
28.cat /tmp/$$ - | $UNCOMPRESS > /dev/null
29.rm /tmp/$$
```

Figure 5.5 downloadClient.sh – script for decoding of incoming file based on ID

Once the download with decompression is completed, the client inserts a new entry into the *historyDecomp* table to describe most recent decompression transaction. The client will calculate compression ratio, CR, and local decompression throughput, $Th.D$, by reversing the model for compressed download with piping (Equation (4.28)). Finally, the uncompressed size (US), compression ratio (CR), and local decompression throughput ($Th.D$), together with corresponding compression utility name (e.g., *xz*) and level (e.g., 9) used for creating the compressed file will be

written back and inserted as a new record in the *historyDecomp* data table on the client. This will refine historical data table with each compressed download transaction, used periodically to update server's *historyDecomp* table for the specific device (e.g., OnePlus One).

$$Th.D = \sqrt{\frac{Th.DW}{\frac{1}{Th.CDW} - \frac{T.SC}{US} - \frac{1}{CR \cdot Th.DW}}} \tag{5.2}$$

5.3 Re-compression and Server's Long-term Storage

5.3.1 Re-compression

To fulfill future download requests from the edge devices, the cloud will have to perform re-compression of incoming files (uploads from the edge devices), and maintain several compressed versions of files in addition to uncompressed files, for cases when downloading the uncompressed file will be more beneficial either with throughput or energy-efficient mode. The information on available versions of each file is kept in the *fileStorage* table. This table is maintained and updated by the server agent with each new upload periodic re-compression, addition, or removal of files.

5.3.2 Optimization

One possible optimization of this feature is to maintain a number of versions of a file based on a parameter such as frequency of usage, download rate or count, or "popularity". This can be specifically appropriate for applications such as application

and software repositories (e.g., Apple Store, Google Play) and distributed storage (e.g., Dropbox).

Taking application repository as an example, for "popular" or "featured" applications that are more likely to be downloaded by a wide variety of users, with a more diverse set of network characteristics, it will be beneficial to maintain multiple compressed files. On the other hand, an application that is not "featured" and not frequently downloaded, can instead maintain 2 or 3 versions of the file that satisfy users with low and high network throughputs (e.g., 3G and WLAN). This way, the framework will able to provide an good throughput and energy efficiency for typical users, while reducing the strain on the server or the cloud, and thus saving the storage space for more frequently downloaded applications instead.

CHAPTER 6

EVALUATION OF FRAMEWORK IMPLEMENTATION

This section covers evaluation of the framework implementations on a smartphone and workstation. Section 6.1 describes the evaluation methods and configurations on the smartphone, workstation, as well as the local and remote cloud instances during framework evaluation. The evaluation results are presented based on the throughput and energy-efficient modes of optimization on the smartphone (Section 6.2), and based on throughput mode of optimization and achieved cost savings of cloud cost on the workstation (Section 6.3).

6.1 Evaluation Methods

6.1.1 Mobile Device

Experimental evaluation of the framework installed on the smartphone involves running upload and download agents on the mobile device and the local server. Each transfer mode selected by the proposed framework is augmented to measure the effective upload and download throughputs and energy efficiencies. To measure effectiveness of the proposed framework with throughput optimized mode, the effective throughputs achieved by the framework agents, *Th.FW*, are compared to the effective throughputs of uncompressed transfers, *Th.UUP [Th.UDW]*, and the default compressed transfers with *gzip* -6, *Th.CUP(gzip -6) [Th.CDW(gzip -6)]*. To measure effectiveness of the proposed framework with energy efficiency optimized

mode, the effective energy efficiencies achieved by the framework agents, *EE.FW*, are compared to the effective energy efficiencies of uncompressed transfers, *EE.UUP [EE.UDW]*, and the default compressed transfers, *EE.CUP(gzip -6) [EE.CDW(gzip -6)]*. The experiments are repeated for WLAN network throughputs set to 0.5, 2, 3.5 and 5 MB/s. The network throughput is set using the Linux *tc* (traffic control) utility. OnePlus One is used as the target device for this evaluation.

Using datasets representative of mobile devices, introduced in Section 2.4.3 (Table 2.3), prediction tables are generated both on the mobile device and copied to the server for all supported compression utilities and levels. This is done by saving compression ratio (*CR*), local throughput (*Th.C*), and uncompressed file size (US) for each executed task into the table. Finally, the number of entries in the prediction tables is reduced and averaged to a certain file size granularity.

6.1.2 Workstation

Experimental evaluation of the framework installed on the workstation involves running upload and download agents on the workstation and the cloud instances in North Virginia and Tokyo. Each transfer mode selected by the proposed framework is augmented to measure the effective upload and download throughputs. To measure effectiveness of the proposed framework, the effective throughputs achieved by the framework agents, *Th.FW*, are compared to the effective throughputs of uncompressed transfers, *Th.UUP [Th.UDW]*, and the default compressed transfers with *gzip* -6, *Th.CUP(gzip -6) [Th.CDW(gzip -6)]*.

In addition to throughput optimization, the effectiveness of the proposed framework is evaluated by the amount of cost savings associated with upload and download to and from the cloud. The total costs for transferring the data with the

framework, *$.FW*, are compared to the total costs with the uncompressed transfers, *$.UUP [$.UDW]*, and the total costs with the default compressed transfers, *$.CUP(gzip -6) [$.CDW(gzip -6)]*.

In preliminary characterization of cloud instances, the high network throughputs are achieved when the user has high bandwidth availability and network priority. However, the network throughput can be capped by the cloud commodity provider (such as AWS) due to high network traffic and low network priority. For this reason, in addition to conducting experiments on the uncapped network, the experiments for uploads and downloads are repeated on two controlled network throughputs. The network throughput is controlled using the Linux *tc* (traffic control) utility to provide controlled network throughputs of 5 MB/s and 2 MB/s locally and on the cloud.

Using datasets representative of workstations, introduced in Section 2.4.3 (Table 2.4), a subset of 20 files is selected from each dataset for conducting experiments. The rest of the files are used to create prediction tables on the workstation and copied to cloud instances for all supported compression utilities and levels. Finally, the number of entries in the prediction tables is reduced and averaged to a certain file size granularity.

6.2 Results: Mobile Device

6.2.1 Throughput Optimized Transfers

This section describes the results of the framework evaluation with throughput optimized file transfers. The experiments are conducted on the OnePlus One smartphone with the 0.5 MB/s, 2 MB/s, 3.5 MB/s, and 5 MB/s WLAN network con-

nections to the local server (Dell PowerEdge T620). Throughput speedups are reported for uploads (Section 6.2.1.1) and downloads (Section 6.2.1.2).

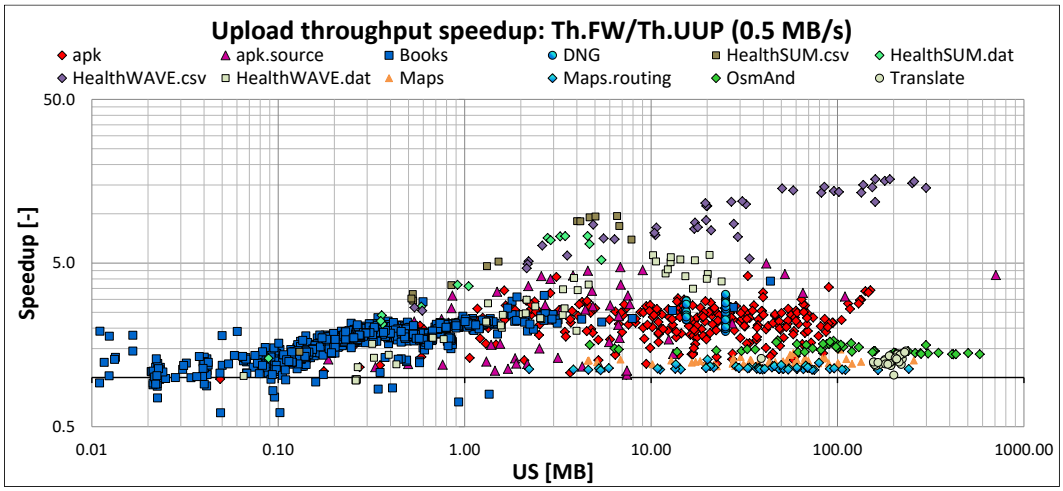
6.2.1.1 Throughput Speedup for Uploads

Th.FW/Th.UUP. The effective upload throughput speedup is calculated as the throughput achieved by the framework divided by the throughput of the corresponding uncompressed upload, *Th.FW/Th.UUP*. Figure 6.1 shows the effective throughput speedups for file transfers when using the 0.5 MB/s (Figure 6.1, a) and the 5 MB/s (Figure 6.1, b) WLAN network. Markers represent speedups for file transfers of a certain type - e.g., red diamonds are apk files and blue squares are book files.

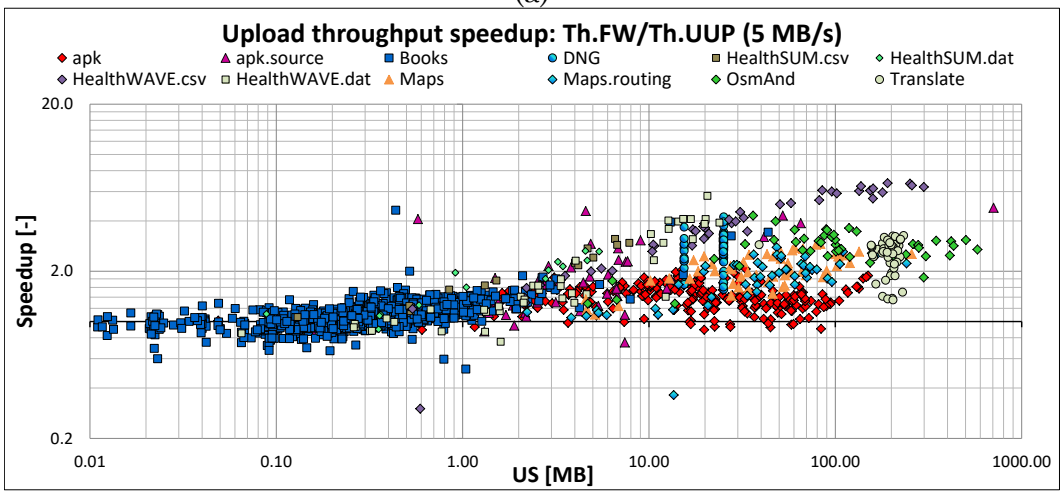
The throughput speedup depends on the file type, the file size, and the network parameters. The optimized file uploads are highly beneficial for the majority of files. Exceptions are relatively small book files. The maximum throughput speedups range from 1.28 for the Maps_routing files to 16.27 for the mHealth waveform files on the 0.5 MB/s network, and from 2.13 for the mHealth binary actively log files to 6.73 for the mHealth waveform files on the 5 MB/s network.

To determine the total throughput speedup achieved by the framework for certain file classes, we divide the sum of total uncompressed transfer times for all files in the group with the sum of total compressed transfer times when using the framework. Table 6.1 shows the total throughput speedups for each file class and for all classes combined (row Total) when transferring files with all selected WLAN network connections. The total throughput speedup for all files ranges from 1.55 (on the 2 MB/s network) to 2.29 (on the 5 MB/s network), being more effective at the higher network throughput. It should be noted that the optimized uploads are most

beneficial for files that are most likely to be uploaded to the cloud, including mHealth files and uncompressed DNG images. The optimized transfers perform well for all network conditions.



(a)



(b)

Figure 6.1 Upload throughput speedup *Th.FW/Th.UUP* on OnePlus One

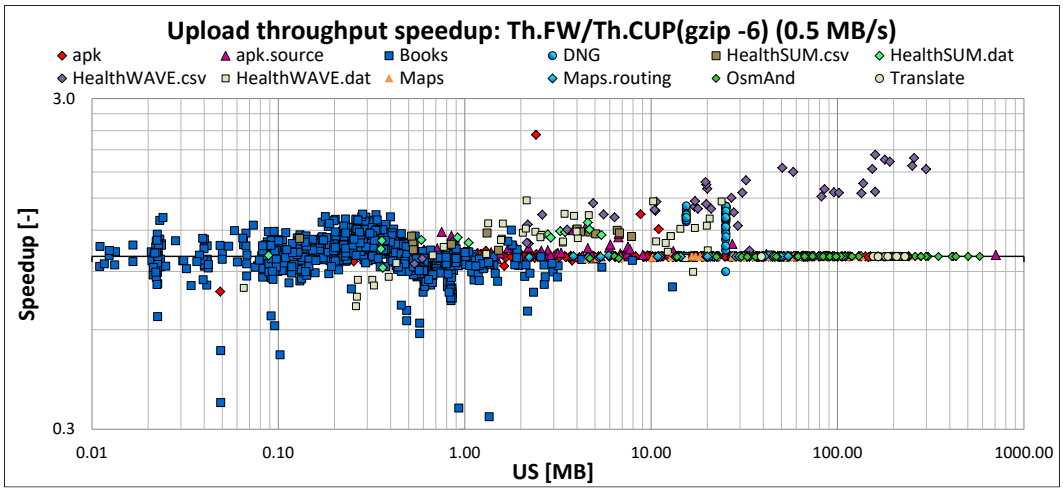
Table 6.1 Overall upload throughput speedup *Th.FW/Th.UUP* on OnePlus One

Th.FW/Th.UUP	0.5 MB/s	2 MB/s	3.5 MB/s	5 MB/s
apk	2.18	2.12	2.07	1.36
apk.source	3.24	2.43	2.75	3.15
Books	1.89	1.30	1.03	1.15
DNG	2.46	1.84	1.73	2.76
HealthSUM.csv	7.24	3.42	2.17	2.17
HealthSUM.dat	5.42	2.76	1.72	1.86
HealthWAVE.csv	13.25	5.81	6.37	5.29
HealthWAVE.dat	3.83	2.53	2.00	2.51
Maps	1.26	1.24	1.23	2.23
Maps.routing	1.14	1.12	1.13	2.03
OsmAnd	1.35	1.33	1.34	2.64
Translate	1.28	1.27	1.28	2.52
Total	**1.55**	**1.49**	**1.44**	**2.29**

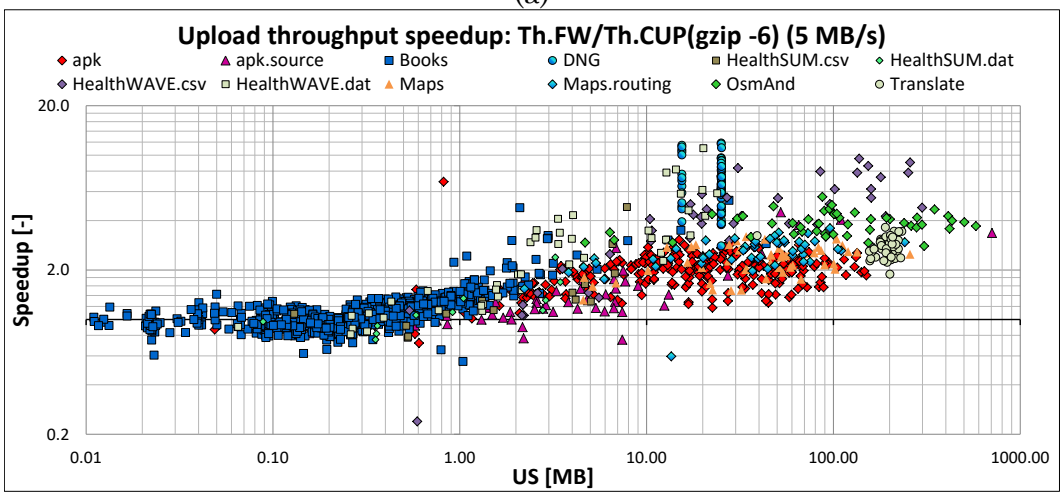
Th.FW/Th.CUP(gzip -6). The effective upload throughput speedup is calculated as the throughput achieved by the framework divided by the throughput of the corresponding default compressed upload, *Th.FW/Th.CUP(gzip -6)*. Figure 6.2 shows the effective upload throughput speedups for file transfers using the 0.5 MB/s (Figure 6.2, a) and the 5 MB/s (Figure 6.2, b) WLAN network.

On the 0.5 MB/s network, the optimized file uploads offer limited speedup for selected files (e.g., DNG, HealthWAVE.dat, and HealthWAVE.csv). Other files have achieved speedup close to 1 or below it, where the framework selects the default compression as its transfer mode. However, even in these conditions, files that are most likely to be uploaded to the cloud benefit from the framework optimizations. The optimized file uploads provide significant improvements over the default compressed transfers on the 5 MB/s network. The maximum throughput speedups range from 2.03 for mHealth waveform text files (on the 0.5 MB/s network) to 12.11 for DNG lossless images (on the 5 MB/s network).

Table 6.2 shows the total throughput speedups for each file class and for all classes combined (row Total) when transferring files with all selected WLAN network connections. The total throughput speedup for all files ranges from 1.01 (on a 0.5 MB/s network) to 2.85 (on a 5 MB/s network).



(a)



(b)

Figure 6.2 Upload throughput speedup *Th.FW/Th.CUP(gzip -6)* on OnePlus One

Table 6.2 Overall upload throughput speedup *Th.FW/Th.CUP(gzip -6)* on OnePlus One

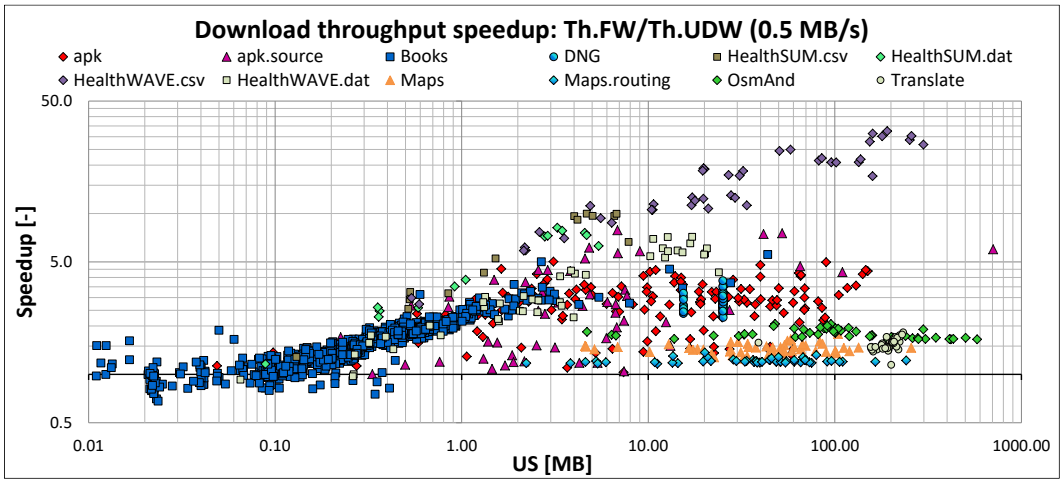
Th.FW/Th.CUP(gzip -6)	0.5 MB/s	2 MB/s	3.5 MB/s	5 MB/s
apk	1.00	1.38	1.29	1.99
apk.source	1.03	1.04	1.62	2.45
Books	0.97	1.11	1.14	1.19
DNG	1.31	2.91	4.50	7.00
HealthSUM.csv	1.16	1.43	1.51	1.80
HealthSUM.dat	1.15	1.57	1.33	1.46
HealthWAVE.csv	1.70	2.65	4.73	5.77
HealthWAVE.dat	1.16	2.31	3.30	3.69
Maps	1.00	1.07	1.61	2.41
Maps.routing	1.00	1.25	1.92	2.71
OsmAnd	1.00	1.24	2.14	3.24
Translate	1.00	1.26	1.84	2.80
Total	**1.01**	**1.31**	**2.01**	**2.85**

6.2.1.2 Throughput Speedup for Downloads

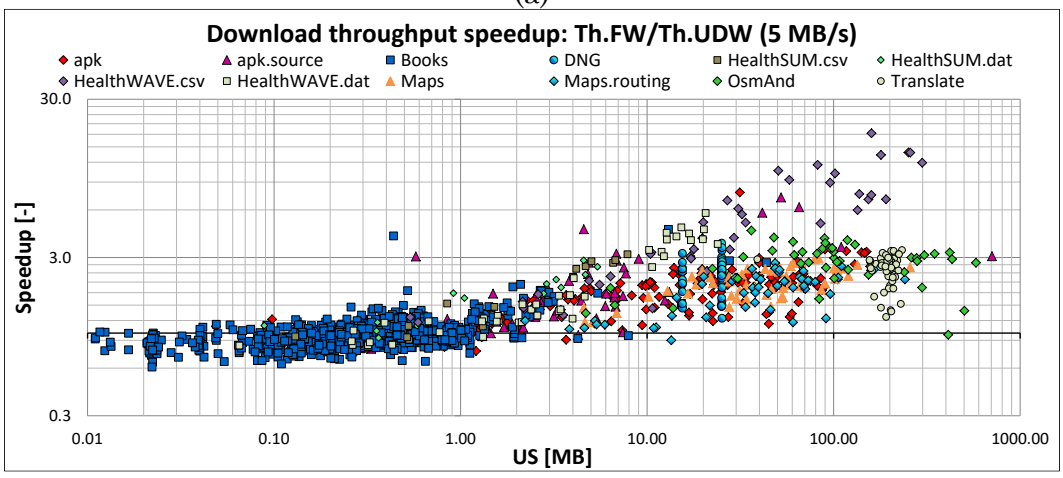
Th.FW/Th.UDW. The effective download throughput speedup is calculated as the throughput achieved by the framework divided by the throughput of the corresponding uncompressed download, *Th.FW/Th.UDW*. Figure 6.3 shows the effective throughput speedups for file transfers when using the 0.5 MB/s (Figure 6.3, a) and the 5 MB/s (Figure 6.3, b) WLAN network.

The optimized file downloads are highly beneficial for the majority of files considered, except for a small group of book files transferred over a high-throughput connection. The maximum throughput speedups range from 1.37 for Maps.routing files to 32.49 for mHealth waveform text files on the 0.5 MB/s network, and from 2.77 for Maps.routing files to 18.28 for mHealth waveform text files on the 5 MB/s network. For transfers on the 0.5 MB/s network, the framework selects utilities with

a high compression ratio, thus increasing the speedup for files larger than 0.1 MB. For transfers on the 5 MB/s network, utilities with higher compression ratio increase speedup for files larger than 1 MB.



(a)



(b)

Figure 6.3 Download throughput speedup *Th.FW/Th.UDW* on OnePlus One

Table 6.3 shows the total throughput speedups for each file class and for all classes combined (row Total) when transferring files with all selected WLAN net-

work connections. The total throughput speedup for all files ranges from 1.78 (on a 0.5 MB/s network) to 2.52 (on a 5 MB/s network), being more effective at the higher network throughput.

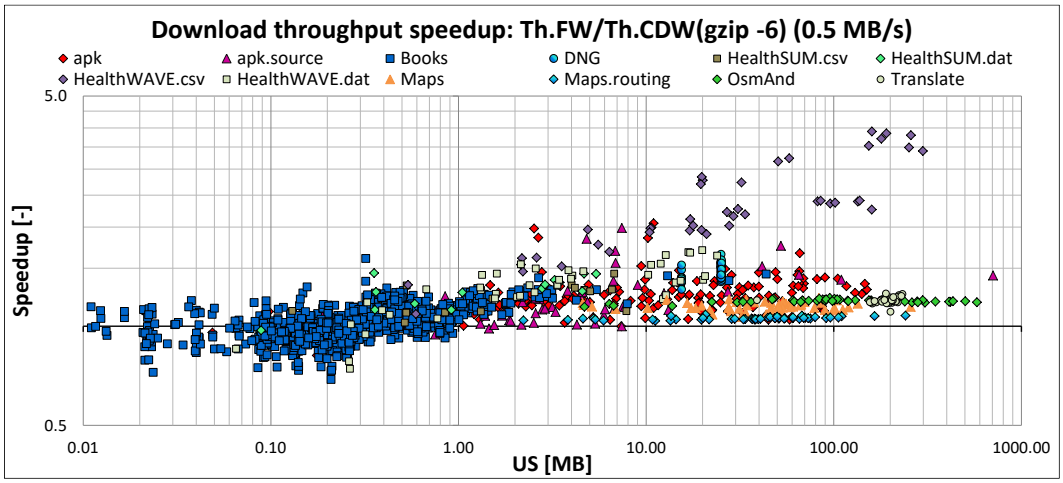
Table 6.3 Overall download throughput speedup *Th.FW/Th.UDW* on OnePlus One

Th.FW/Th.UDW	0.5 MB/s	2 MB/s	3.5 MB/s	5 MB/s
apk	2.73	2.87	2.44	1.97
apk.source	2.88	2.69	2.16	2.18
Books	1.92	1.20	1.03	1.01
DNG	2.78	2.12	1.28	2.35
HealthSUM.csv	7.11	3.62	2.28	2.05
HealthSUM.dat	5.72	2.84	1.92	1.83
HealthWAVE.csv	22.18	15.68	9.16	7.94
HealthWAVE.dat	4.58	3.00	2.29	2.68
Maps	1.45	1.44	1.25	2.27
Maps.routing	1.21	1.15	1.15	2.02
OsmAnd	1.63	1.60	1.36	2.51
Translate	1.55	1.53	1.29	2.55
Total	**1.78**	**1.72**	**1.44**	**2.52**

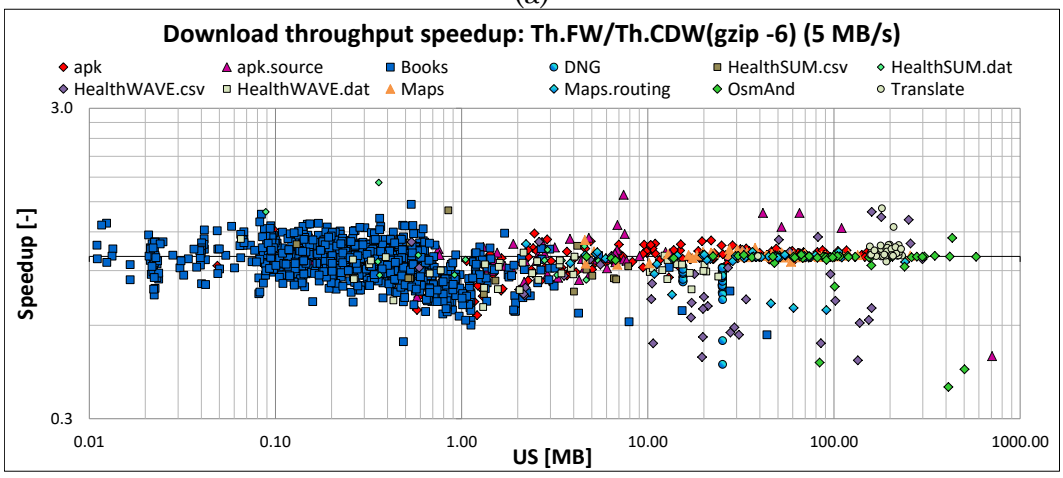
Th.FW/Th.CDW(gzip -6). The effective download throughput speedup is calculated as the throughput achieved by the framework divided by the throughput of the corresponding default compressed download, *Th.FW/Th.CDW(gzip -6)*. Figure 6.4 shows the effective throughput speedups for file transfers when using the 0.5 MB/s (Figure 6.4, a) and the 5 MB/s (Figure 6.4, b) WLAN network.

On the 0.5 MB/s network, the optimized file downloads are beneficial except for small book files, where the framework selects the default compressed file from the server during its transfer. The optimized file downloads for high network

throughput (5 MB/s), offers limited speedup, with many file transfers achieving a speedup of 1 or below it. The maximum throughput speedups range from 1.08 for Maps.routing files to 3.90 for mHealth waveform text files on the 0.5 MB/s network.



(a)



(b)

Figure 6.4 Download throughput speedup *Th.FW/Th.CDW(gzip -6)* on OnePlus One

Table 6.4 shows the total throughput speedups for each file class and for all classes combined (row Total) when transferring files with all selected WLAN network connections. The total throughput speedup for all files ranges from 1 (on the 5 MB/s network) to 1.2 (on the 0.5 MB/s network), being more effective at the lower network throughput. The results show that optimized downloads outperform the default compressed downloads at lower network throughputs, but offer no significant advantages at the higher network throughputs.

Table 6.4 Overall download throughput speedup *Th.FW/Th.CDW(gzip -6)* on OnePlus One

Th.FW/Th.CDW(gzip -6)	0.5 MB/s	2 MB/s	3.5 MB/s	5 MB/s
apk	1.24	1.22	1.15	0.96
apk.source	1.25	1.22	1.11	1.02
Books	1.17	1.24	0.96	1.12
DNG	1.47	1.27	1.15	0.85
HealthSUM.csv	1.21	1.08	1.08	0.92
HealthSUM.dat	1.29	1.13	0.97	1.02
HealthWAVE.csv	2.84	2.13	1.32	0.84
HealthWAVE.dat	1.40	1.10	0.96	0.90
Maps	1.15	1.15	0.99	1.01
Maps.routing	1.06	1.01	1.00	0.96
OsmAnd	1.20	1.18	1.00	0.98
Translate	1.20	1.19	1.00	1.04
Total	**1.20**	**1.18**	**1.01**	**1.00**

6.2.2 Energy Efficiency Optimized Transfers

This section describes the results of the framework evaluation with energy efficiency optimized file transfers. The experiments are conducted on the OnePlus

One smartphone with the 0.5 MB/s, 2 MB/s, 3.5 MB/s, and 5 MB/s WLAN network connections to the local server (Dell PowerEdge T620). Improvements in energy efficiency are reported for uploads (Section 6.2.2.1) and downloads (Section 6.2.2.1).

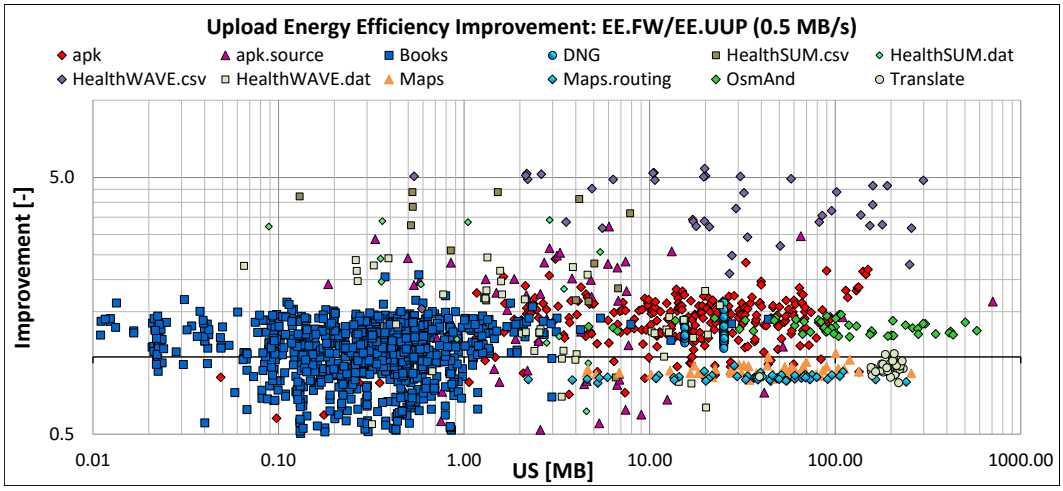
6.2.2.1 Energy Efficiency Improvement for Uploads

EE.FW/EE.UUP. The effective upload energy efficiency improvement is calculated as the energy efficiency achieved by the framework divided by the energy efficiency of the corresponding uncompressed upload, *EE.FW/EE.UUP*. Figure 6.5 shows the effective energy efficiency improvements for file transfers when using the 0.5 MB/s (Figure 6.5, a) and the 5 MB/s (Figure 6.5, b) WLAN network.

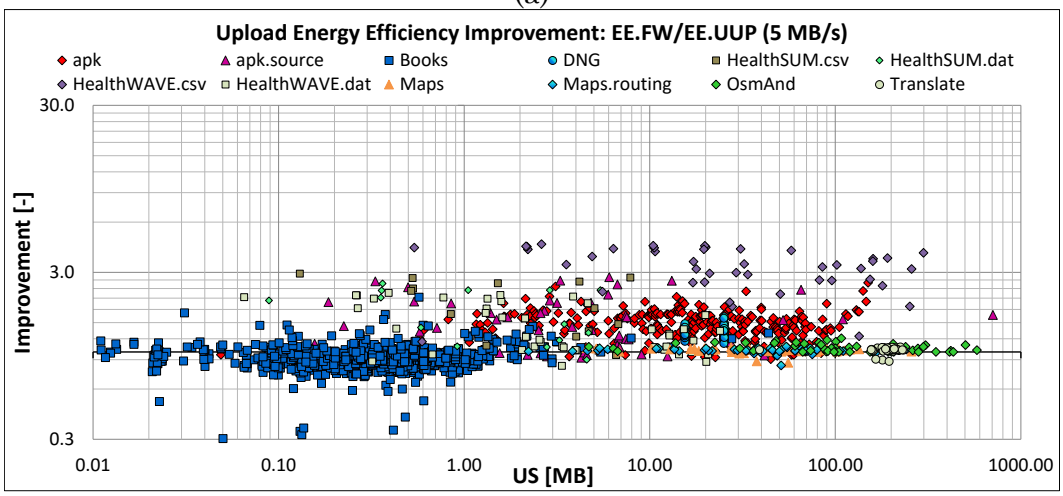
The energy efficiency improvement depends on the file type, the file size, and the network parameters. On the 0.5 MB/s network, the optimized file uploads offer good optimization for the majority of selected files. However, on the 5 MB/s network, files that are less than 1 MB in size and low compressible files, have achieved energy efficiency improvement close to 1 or below it, where the framework selects the uncompressed transfer mode. Energy efficiency is improved for the majority of larger files with higher compression ratio.

To determine the total energy efficiency improvement achieved by the framework for certain file classes, we divide the sum of total energy of uncompressed transfer for all files in the group with the sum of total energy when using the framework. Table 6.5 shows the total energy efficiency improvements for each file class and for all classes combined (row Total) when transferring files with all selected WLAN network connections. The total energy efficiency improvement for all files ranges from 1.09 (on the 0.5 MB/s network) to 1.16 (on the 5 MB/s network). It should be noted that the optimized uploads are most beneficial for files that are most

likely to be uploaded to the cloud, including mHealth files and uncompressed DNG images.



(a)



(b)

Figure 6.5 Upload energy efficiency improvement *EE.FW/EE.UDW* on OnePlus One

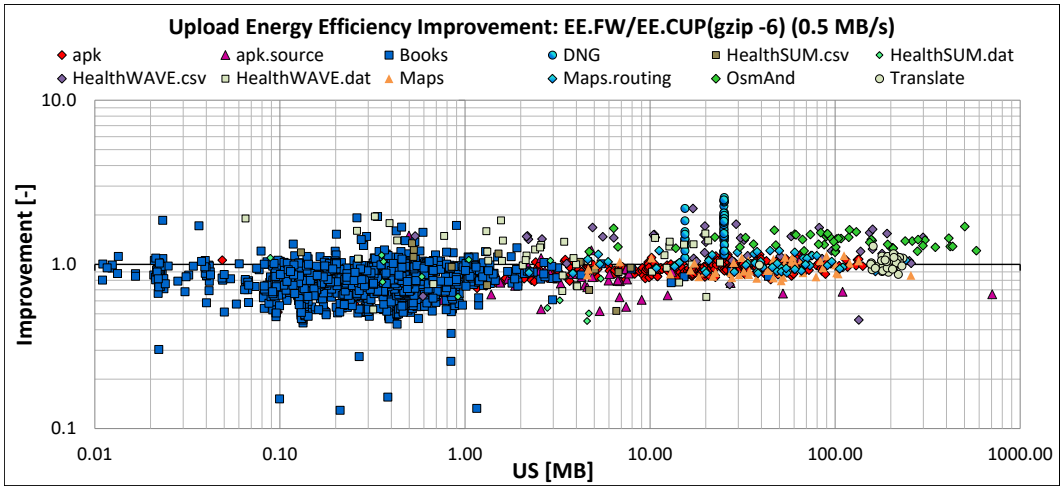
Table 6.5 Overall upload energy efficiency improvement *EE.FW/EE.UUP* on OnePlus One

EE.FW/EE.UUP	0.5 MB/s	2 MB/s	3.5 MB/s	5 MB/s
apk	1.42	1.62	1.60	1.44
apk.source	1.14	2.02	2.12	2.03
Books	1.17	0.94	0.93	0.97
DNG	1.25	1.32	1.38	1.29
HealthSUM.csv	3.42	2.54	2.60	2.25
HealthSUM.dat	2.37	2.32	2.15	1.93
HealthWAVE.csv	4.76	4.24	4.40	3.97
HealthWAVE.dat	1.84	1.93	1.93	1.75
Maps	0.89	0.96	1.02	1.03
Maps.routing	0.86	0.95	1.02	1.03
OsmAnd	1.04	1.04	1.06	1.03
Translate	0.91	0.98	1.06	1.03
Total	**1.09**	**1.16**	**1.20**	**1.16**

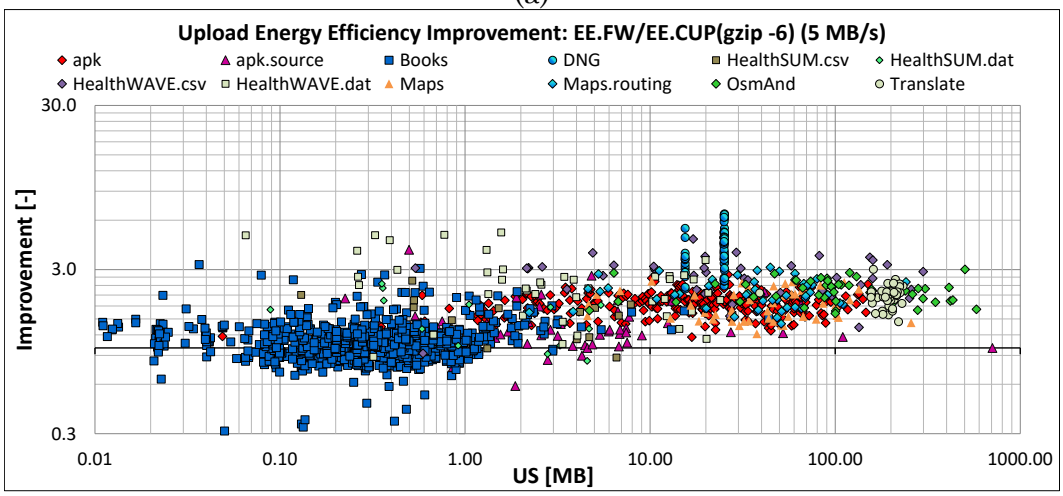
EE.FW/EE.CUP(gzip -6). The effective upload energy efficiency improvement is calculated as the energy efficiency achieved by the framework divided by the energy efficiency of the corresponding default compressed upload, *EE.FW/EE.CUP(gzip -6)*. Figure 6.6 shows the effective energy efficiency improvement for file transfers when using the 0.5 MB/s (Figure 6.6, a) and the 5 MB/s (Figure 6.6, b) WLAN network.

On the 0.5 MB/s network, the optimized file uploads offer limited improvement for selected files (e.g., DNG, HealthSUM.csv, HealthWAVE.csv, and HealthWAVE.dat). Other files, including files that are less than 1 MB in size, have achieved energy efficiency improvement close to 1 or below it, where the framework selects the default compression as its transfer mode. The optimized file uploads provide significant improvements over the default compressed transfers on the 5 MB/s

network. Overall, files that are most likely to be uploaded to the cloud benefit from the framework optimizations.



(a)



(b)

Figure 6.6 Upload energy efficiency improvement *EE.FW/EE.CUP(gzip -6)* on OnePlus One

Table 6.6 shows the total energy efficiency improvements for each file class and for all classes combined (row Total) when transferring files with all selected

WLAN network connections. The total energy efficiency improvement for all files ranges from 1.09 (on the 0.5 MB/s network) to 2.13 (on the 5 MB/s network), being more effective at the higher network throughput.

Table 6.6 Overall upload energy efficiency improvement *EE.FW/EE.CUP(gzip -6)* on OnePlus One

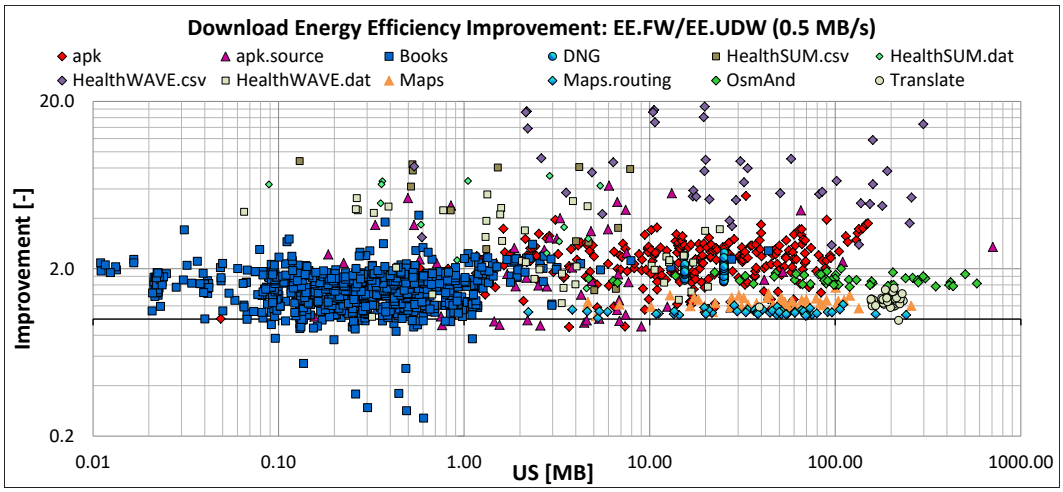
EE.FW/EE.CUP(gzip -6)	0.5 MB/s	2 MB/s	3.5 MB/s	5 MB/s
apk	0.96	1.50	1.75	1.84
apk.source	0.78	1.28	1.57	1.74
Books	0.86	0.94	1.08	1.20
DNG	1.66	2.82	3.62	4.02
HealthSUM.csv	1.10	1.32	1.71	1.81
HealthSUM.dat	0.90	1.39	1.61	1.75
HealthWAVE.csv	1.46	2.11	2.77	3.08
HealthWAVE.dat	1.37	2.38	2.95	3.15
Maps	0.97	1.36	1.65	1.91
Maps.routing	1.01	1.53	1.90	2.18
OsmAnd	1.15	1.61	1.94	2.15
Translate	1.02	1.50	1.90	2.11
Total	**1.09**	**1.58**	**1.93**	**2.13**

6.2.2.2 Energy Efficiency Improvement for Downloads

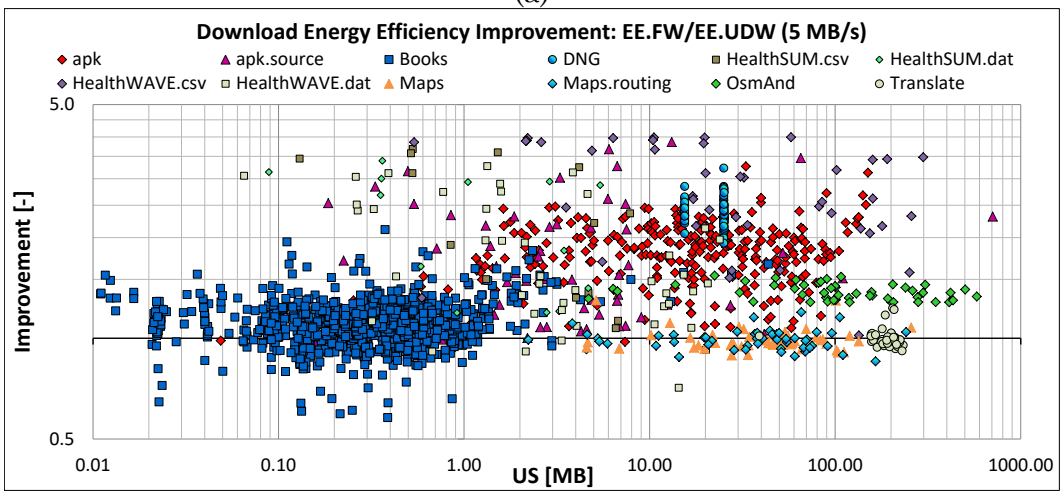
EE.FW/EE.UDW. The effective download energy efficiency improvement is calculated as the energy efficiency achieved by the framework divided by the energy efficiency of the corresponding uncompressed download, *EE.FW/EE.UDW*. Figure 6.7 shows the effective energy efficiency improvements for file transfers when using the 0.5 MB/s (Figure 6.7, a) and the 5 MB/s (Figure 6.7, b) WLAN network.

The optimized file downloads are highly beneficial for the majority of files considered. On the 0.5 MB/s network, the optimized file downloads offer good optimization for the majority of selected files. On the 5 MB/s network, the optimizations offer a limited improvement in energy efficiency for the majority of files smaller than 1 MB (e.g., Books). Energy efficiency is improved for the majority of larger files with higher compression ratio.

Table 6.7 shows the total energy efficiency improvements for each file class and for all classes combined (row Total) when downloading files with all selected WLAN network connections. The total energy efficiency improvement for all files ranges from 1.45 (on the 0.5 MB/s network) to 1.24 (on the 5 MB/s network), being more effective at the lower network throughput. It should be noted that the optimized transfers of files with medium to high compression ratios (e.g., apk, apksource, and Books) are most beneficial with all network connections, while the optimized transfers of files with low compression ratios (e.g, Maps, Maps.routing, OsmAnd, and Translate) are beneficial with the low network connections.



(a)



(b)

Figure 6.7 Download energy efficiency improvement *EE.FW/EE.UDW* on OnePlus One

Table 6.7 Overall download energy efficiency improvement *EE.FW/EE.UDW* on OnePlus One

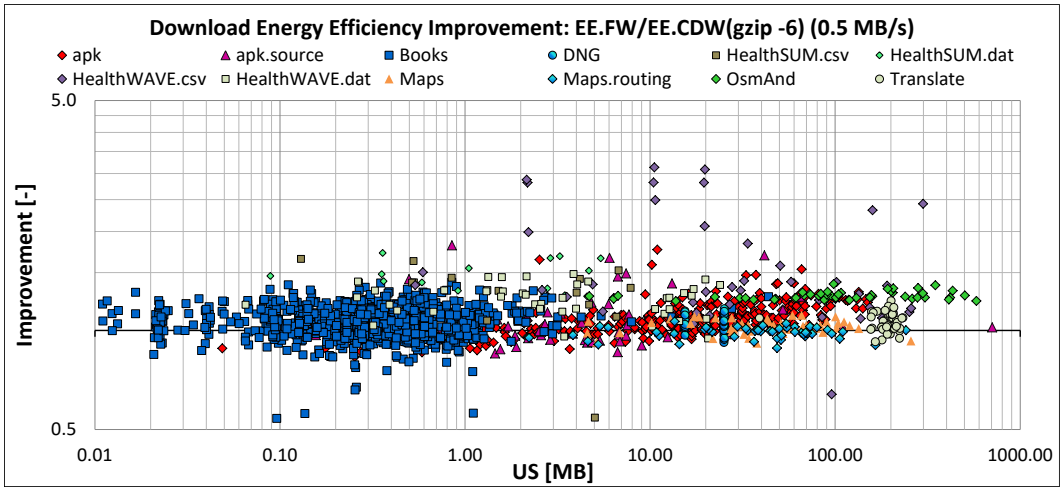
EE.FW/EE.UDW	0.5 MB/s	2 MB/s	3.5 MB/s	5 MB/s
apk	2.00	1.84	1.61	1.80
apk.source	1.68	1.66	1.47	2.50
Books	1.71	1.53	1.32	1.10
DNG	1.93	2.40	2.26	2.30
HealthSUM.csv	5.88	4.08	2.98	2.64
HealthSUM.dat	5.27	3.08	2.41	2.32
HealthWAVE.csv	10.77	11.20	10.20	3.51
HealthWAVE.dat	3.58	2.71	2.13	2.16
Maps	1.25	1.12	1.01	0.99
Maps.routing	1.10	1.01	1.00	0.99
OsmAnd	1.47	1.25	1.12	1.12
Translate	1.35	1.12	1.01	1.00
Total	**1.51**	**1.37**	**1.24**	**1.24**

EE.FW/EE.CDW(gzip -6). The effective download energy efficiency improvement is calculated as the energy efficiency achieved by the framework divided by the energy efficiency of the corresponding default compressed download, *EE.FW/EE.CDW(gzip -6)*. Figure 6.8 shows the effective energy efficiency improvement for file transfers when using the 0.5 MB/s (Figure 6.8, a) and the 5 MB/s (Figure 6.8, b) WLAN network.

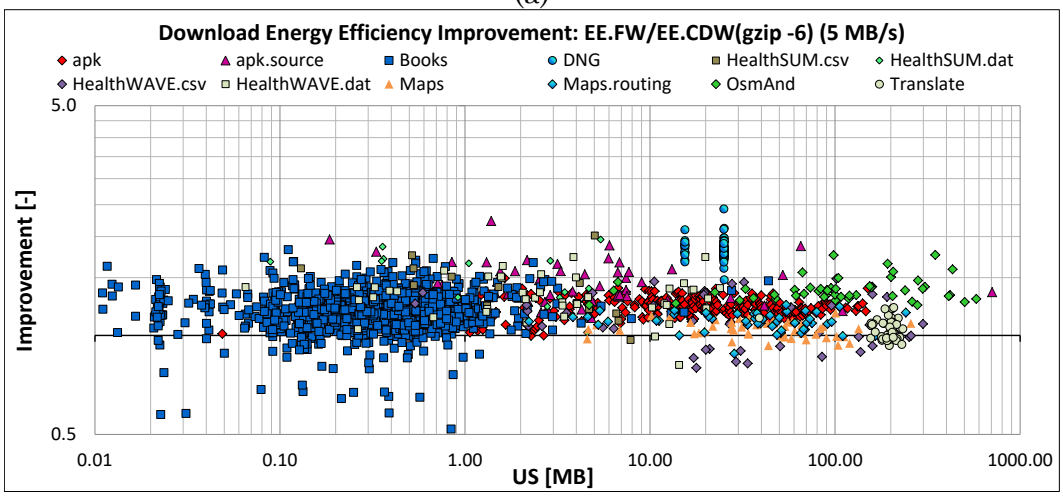
On the 0.5 MB/s network, the optimizations offer a limited improvement in energy efficiency for the majority of files with low compressibility (e.g., Maps, Maps.routing). On the 5 MB/s network, the optimized file downloads are highly beneficial for the majority of files in the dataset.

Table 6.8 shows the total energy efficiency improvements for each file class and for all classes combined (row Total) when downloading files with all selected WLAN network connections. The total energy efficiency improvement for all files

ranges from 1.12 (on the 0.5 MB/s network) to 1.17 (on the 5 MB/s network), being more effective at the higher network throughput.



(a)



(b)

Figure 6.8 Download energy efficiency improvement *EE.FW/EE.CDW(gzip -6)* on OnePlus One

Table 6.8 Overall download energy efficiency improvement *EE.FW/EE.CDW(gzip -6)* on OnePlus One

EE.FW/EE.CUP(gzip -6)	0.5 MB/s	2 MB/s	3.5 MB/s	5 MB/s
apk	1.04	1.03	1.04	1.21
apk.source	1.05	1.19	1.27	1.44
Books	1.04	1.15	1.20	1.13
DNG	1.06	1.52	1.65	1.84
HealthSUM.csv	1.29	1.28	1.34	1.42
HealthSUM.dat	1.51	1.31	1.37	1.56
HealthWAVE.csv	1.84	2.36	2.90	1.17
HealthWAVE.dat	1.29	1.18	1.18	1.33
Maps	1.05	1.03	1.04	1.06
Maps.routing	1.01	1.01	1.12	1.13
OsmAnd	1.18	1.02	1.04	1.22
Translate	1.09	1.00	1.03	1.06
Total	**1.12**	**1.04**	**1.09**	**1.17**

6.3 Results: Workstation

6.3.1 Throughput Optimized Transfers

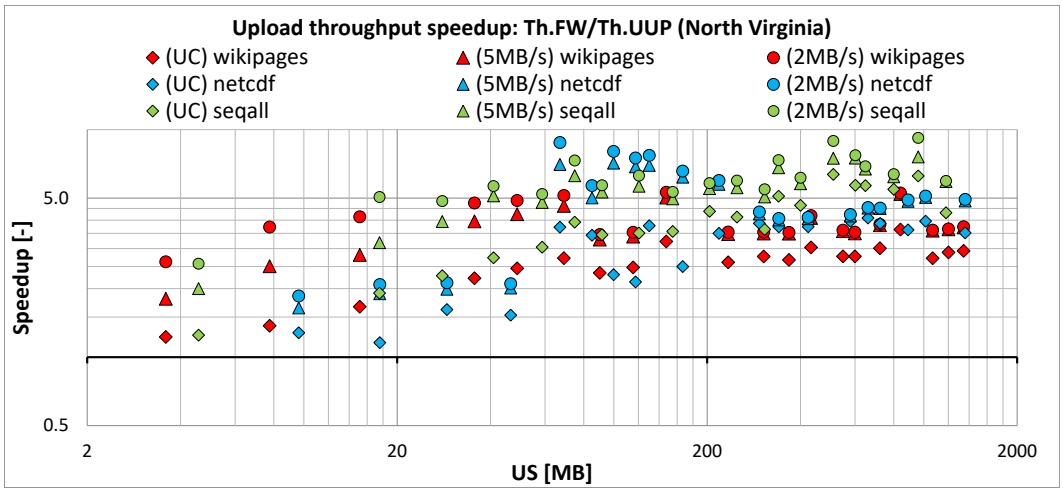
This section describes the results of the framework evaluation with throughput optimized file transfers. The experiments are conducted on the Dell PowerEdge T110 II machine with the uncapped, 5 MB/s and 2 MB/s LAN network connections to the cloud instances in North Virginia and Tokyo. Throughput speedups are reported for uploads (Section 6.3.1.1) and downloads (Section 6.3.1.2).

6.3.1.1 Throughput Speedup for Uploads

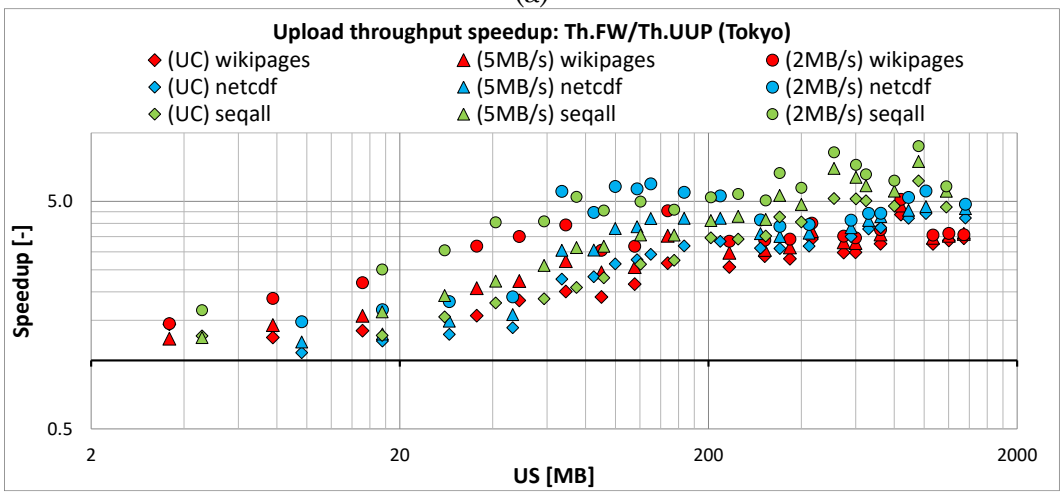
Th.FW/Th.UUP. The effective upload throughput speedup is calculated as the throughput achieved by the framework divided by the throughput of the corre-

sponding uncompressed upload, *Th.FW/Th.UUP*. Figure 6.9 shows the effective throughput speedups for transfers to the North Virginia (Figure 6.9, a) and the Tokyo (Figure 6.9, b) cloud instances. Markers represent speedups for file transfers over the uncapped network (diamond), the 5 MB/s LAN (triangle), and 2 MB/s LAN (circle). Marker colors represent three types of the selected datasets –the red markers are used for *wikipages* files, the blue for *netcdf* files, and green for *seqall* files.

The throughput speedup depends on the file type, the file size, and the network parameters. The optimized file uploads are highly beneficial for all files. For transfers to the North Virginia instance, the throughput speedups range from 1.20 to 6.25 with the uncapped network, 1.80 to 7.58 with the 5 MB/s network, and 2.62 to 9.19 with the 2 MB/s network. For transfers to the Tokyo instance, the throughput speedups range from 1.08 to 6.15 with the uncapped network, 1.24 to 7.48 with the 5 MB/s network, and 1.45 to 8.74 with the 2 MB/s network.



(a)



(b)

Figure 6.9 Upload throughput speedup *Th.FW/Th.UUP*:

North Virginia (a) and Tokyo (b)

To determine the total throughput speedup achieved by the framework for certain file classes, we divide the sum of the total uncompressed transfer times for all files in the dataset with the sum of the total compressed transfer times when using the framework. Table 6.9 and Table 6.10 show the total throughput speedups for each file class and for all classes combined (row Total) when transferring files to the North Virginia and Tokyo clouds, respectively. The total throughput speedups for all

files range from 3.51 (on the uncapped network) to 4.76 (on the 2 MB/s network) for the North Virginia cloud, and from 3.43 to 4.54 for the Tokyo cloud. In both cases, speedups are greater on the lower network throughput.

Table 6.9 Overall upload throughput speedup *Th.FW/Th.UUP* (North Virginia)

Th.FW/Th.UUP	Uncapped	5 MB/s	2 MB/s
wikipages	2.86	3.78	3.83
netcdf	3.52	4.64	4.74
seqall	4.78	6.28	6.74
Total	**3.51**	**4.62**	**4.76**

Table 6.10 Overall upload throughput speedup *Th.FW/Th.UUP* (Tokyo)

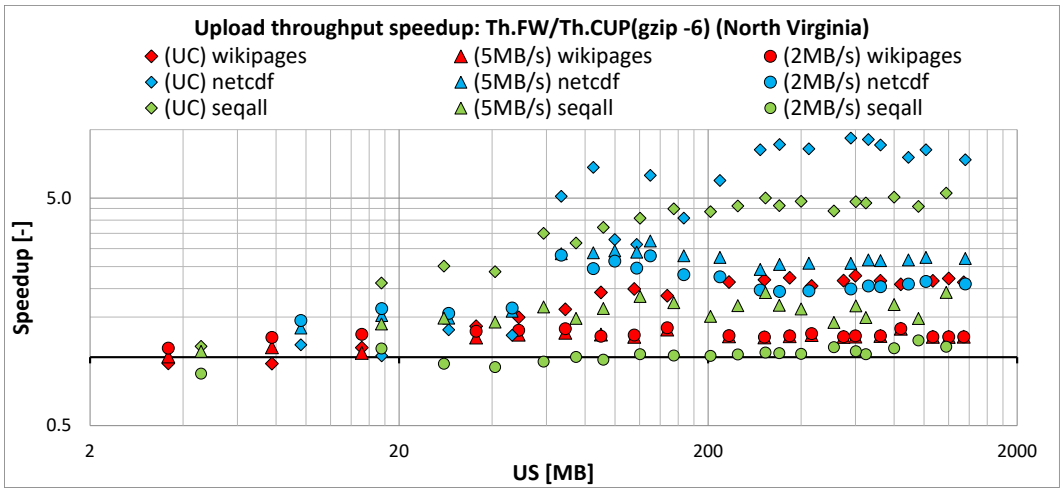
Th.FW/Th.UUP	Uncapped	5 MB/s	2 MB/s
wikipages	3.05	3.41	3.67
netcdf	3.43	4.00	4.61
seqall	4.02	5.09	6.18
Total	**3.43**	**4.01**	**4.54**

Th.FW/Th.CUP(gzip -6). The effective upload throughput speedup is calculated as the throughput achieved by the framework divided by the throughput of the corresponding default compressed upload with *gzip* -6, *Th.FW/Th.CUP(gzip* -6*).* Figure 6.10 shows the effective throughput speedups for transfers to the North Virginia (Figure 6.10, a) and the Tokyo (Figure 6.10, b) cloud instances.

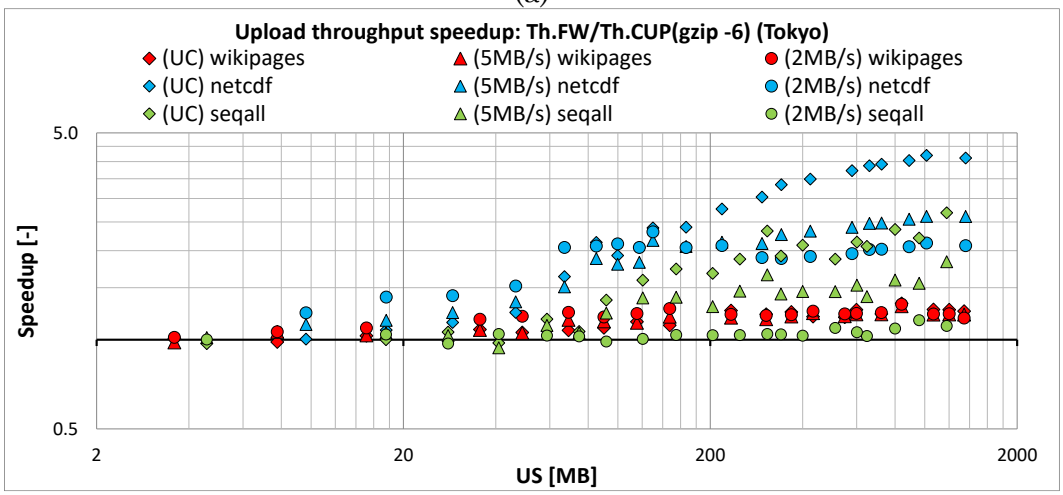
The optimized file uploads are highly beneficial for all files, except when transferring *seqall* dataset on the 2 MB/s network connection, which achieve speedups close to one. In this case, the framework selects the default compression or similarly performing compression (utility, level) pair to perform the upload. For

transfers to the North Virginia instance, the throughput speedups range from 0.94 to 9.04 with the uncapped network, 1.0 to 2.71 with the 5 MB/s network, and 1.09 to 2.10 with the 2 MB/s network. For transfers to the Tokyo instance, the throughput speedups range from 1.01 to 4.11 with the uncapped network, 0.98 to 2.61 with the 5 MB/s network, and 1.01 to 2.08 with the 2 MB/s network.

Table 6.11 and Table 6.12 show the total throughput speedups for each file class and for all classes combined (row Total) when transferring files to the North Virginia and Tokyo clouds, respectively. The total throughput speedup for all files used in the evaluation ranges from 4.36 (on the uncapped network) to 1.48 (on the 2 MB/s network) for the North Virginia cloud, and from 2.10 to 1.44 for the Tokyo cloud. In both cases, speedups are greater at the higher network throughput.



(a)



(b)

Figure 6.10 Upload throughput speedup *Th.FW/Th.CUP(gzip -6)*:

North Virginia (a) and Tokyo (b)

Table 6.11 Overall upload throughput speedup *Th.FW/Th.CUP(gzip -6)*

(North Virginia)

Th.FW/Th.CUP(gzip -6)	Uncapped	5 MB/s	2 MB/s
wikipages	2.12	1.24	1.24
netcdf	7.25	2.63	2.07
seqall	4.63	1.67	1.07
Total	**4.36**	**1.79**	**1.48**

Table 6.12 Overall upload throughput Speedup *Th.FW/Th.CUP(gzip -6)* (Tokyo)

Th.FW/Th.CUP(gzip -6)	Uncapped	5 MB/s	2 MB/s
wikipages	1.22	1.21	1.22
netcdf	3.32	2.32	2.00
seqall	1.97	1.48	1.07
Total	**2.10**	**1.64**	**1.44**

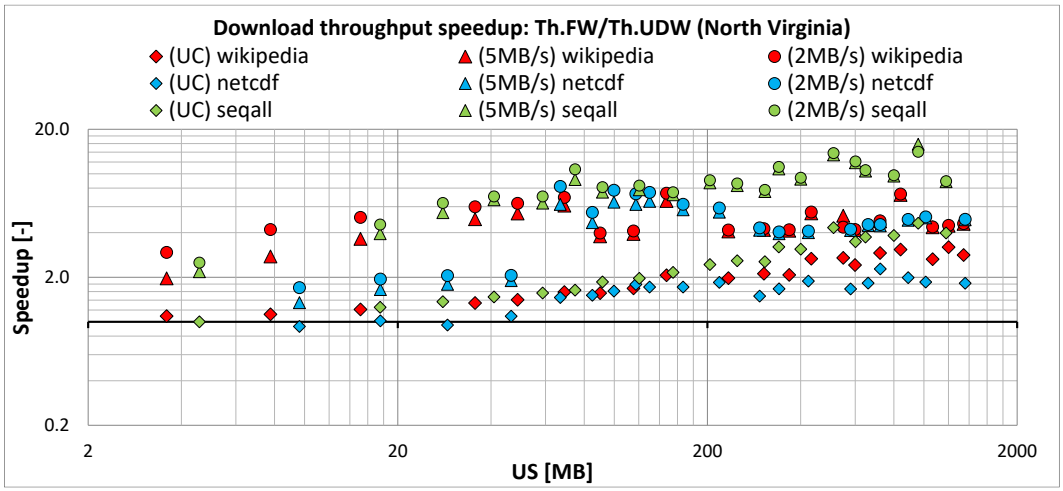
6.3.1.2 Throughput Speedup for Downloads

Th.FW/Th.UDW. The effective download throughput speedup is calculated as the throughput achieved by the framework divided by the throughput of the corresponding uncompressed download, *Th.FW/Th.UDW*. Figure 6.11 shows the effective throughput speedups for transfers from the North Virginia (Figure 6.11, a) and the Tokyo (Figure 6.11, b) cloud instances.

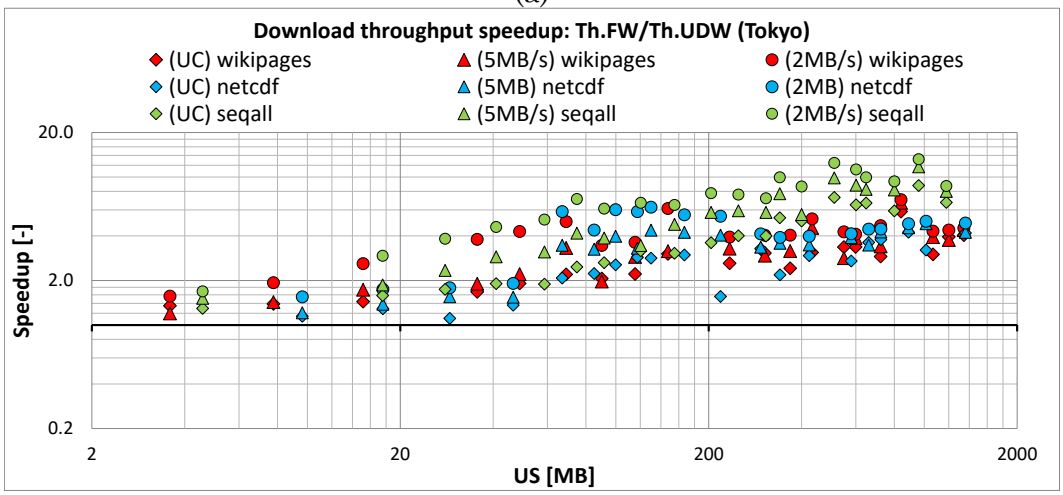
The optimized file downloads are highly beneficial for all files, except for smaller files on the uncapped network, which achieve speedups close to one. In this case, the framework selects uncompressed transfer to perform the download. For transfers from the North Virginia instance, the throughput speedups range from 1.09 to 4.63 with the uncapped network, 1.97 to 15.80 with the 5 MB/s network, and 1.7 to 14.07 with the 2 MB/s network. For transfers from the Tokyo instance, the throughput speedups range from 3.59 to 8.76 with the uncapped network, 1.2 to 11.72 with the 5 MB/s network, and 1.57 to 13.20 with the 2 MB/s network.

Table 6.13 and Table 6.14 show the total throughput speedups for each file class and for all classes combined (row Total) when transferring files from the North Virginia and Tokyo clouds, respectively. The total throughput speedups range from

2.3 (on the uncapped network) to 5.64 (on the 2 MB/s network) for the North Virginia cloud, and from 3.52 to 5.32 for the Tokyo cloud. In both cases, speedups are greater on connections with lower network throughput.



(a)



(b)

Figure 6.11 Download throughput speedup *Th.FW/Th.UDW*: North Virginia (a) and Tokyo (b)

Table 6.13 Overall download throughput speedup *Th.FW/Th.UDW* (North Virginia)

Th.FW/Th.UDW	Uncapped	5 MB/s	2 MB/s
wikipages	2.49	4.72	4.73
netcdf	1.76	4.57	4.72
seqall	3.03	10.00	10.09
Total	**2.30**	**5.56**	**5.64**

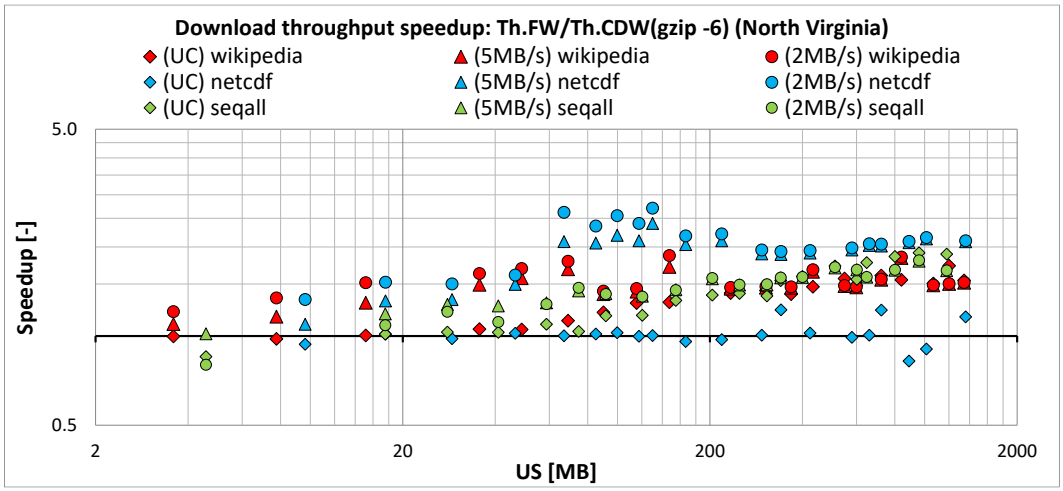
Table 6.14 Overall download throughput speedup *Th.FW/Th.UDW* (Tokyo)

Th.FW/Th.UDW	Uncapped	5 MB/s	2 MB/s
wikipages	3.22	3.69	4.52
netcdf	3.01	3.88	4.53
seqall	4.95	6.97	9.08
Total	**3.52**	**4.40**	**5.35**

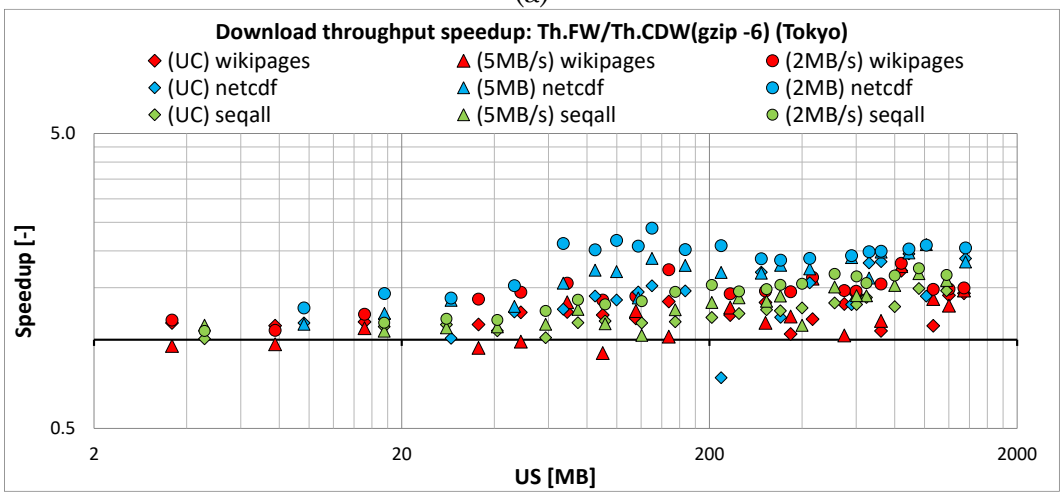
Th.FW/Th.CDW(gzip -6). The effective download throughput speedup is calculated as the throughput achieved by the framework divided by the throughput of the corresponding default compressed download, *Th.FW/Th.CDW(gzip -6)*. Figure 6.12 shows the effective throughput speedups for transfers from the North Virginia (Figure 6.12, a) and the Tokyo (Figure 6.12, b) cloud instances.

The optimized file downloads are highly beneficial for all files, except when transferring *netcdf* files and small files on the uncapped network connection, which achieve speedups close to one. In this case, the framework selects the default compression or similarly performing compression (utility, level) pair to perform the download. For transfers from the North Virginia cloud, the throughput speedups range from 1.0 to 1.89 with the uncapped network, 1.1 to 2.08 with the 5 MB/s network, and 1.21 to 2.10 with the 2 MB/s network. For transfers from the Tokyo instance, the throughput speedups range from 1.01 to 1.95 with the uncapped net-

work, 0.95 to 2.10 with the 5 MB/s network, and 1.17 to 2.05 with the 2 MB/s network.



(a)



(b)

Figure 6.12 Download throughput speedup *Th.FW/Th.CDW(gzip -6)*: North Virginia (a) and Tokyo (b)

Table 6.15 and Table 6.16 show the total throughput speedups for each file class and for all classes combined (row Total) when transferring files from the North Virginia and Tokyo clouds, respectively. The total throughput speedups for all files

range from 1.29 (on the uncapped network) to 1.75 (on the 2 MB/s network) for the North Virginia cloud, and from 1.36 to 1.70 for the Tokyo cloud. In both cases, speedups are greater at the lower network throughput.

Table 6.15 Overall download throughput speedup *Th.FW/Th.CDW(gzip -6)* (North Virginia)

Th.FW/Th.CDW(gzip -6)	Uncapped	5 MB/s	2 MB/s
wikipages	1.46	1.52	1.53
netcdf	1.02	2.01	2.06
seqall	1.52	1.60	1.61
Total	**1.29**	**1.73**	**1.75**

Table 6.16 Overall download throughput speedup *Th.FW/Th.CDW(gzip -6)* (Tokyo)

Th.FW/Th.CDW(gzip -6)	Uncapped	5 MB/s	2 MB/s
wikipages	1.28	1.30	1.51
netcdf	1.50	1.80	1.98
seqall	1.27	1.38	1.56
Total	**1.36**	**1.50**	**1.70**

6.3.2 Cost Savings

The optimized file transfers can also result in a reduction of costs associated with the cloud platforms such as Amazon's AWS EC2. For example, the m4.xlarge cloud instance type used in the North Virginia and Tokyo clouds has the on-demand utilization cost of $0.239 and $0.348 per hour, respectively. Additionally, the on-demand pricing includes the cost for transferring data out from the cloud. For the North Virginia cloud, it is priced at $0.09 per GB (for the first 10 TB per month). For the Tokyo cloud, it is priced at $0.14 per GB. Transfers to and between cloud in-

stances in the same region are free; however, the on-demand utilization fees apply. To optimize utilization of the cloud for computational offload, a typical usage scenario would include following steps: (i) start a cloud instance (initiating the on-demand charge), (ii) transfer of the files needed for completion of a certain task, (iii) execute the task, (iv) download of the results, and finally (v) shut down the instance (ending the on-demand charge).

To calculate the cost savings, first, the total costs of transferring data are calculated using Equations (6.1), (6.2), and (6.3). The total cost of uploading a file to the cloud using uncompressed, default compressed, and framework compressed transfer mode depends on the utilization fee, $Util_{FEE}$, and the execution time of the upload (converted to hours from seconds), as shown in Equation (6.1). The optimal cost is achieved by a transfer mode with the highest upload throughput. The total cost of downloading an uncompressed file depends on the utilization fee, the execution time of download (converted to hours from seconds), the transfer out fee, $Dout_{FEE}$, and the uncompressed file size (converted to GB from MB), as shown in Equation (6.2). For the default compressed and the framework compressed transfer mode, the download out portion of the total cost is lower due to transferring the compressed file, as shown in Equation (6.3). The optimal download cost depends on the balance between highest throughput and highest compression ratio.

$$\$.UUP[\$.CUP][\$.FW] = \frac{Util_{FEE} \cdot T.UUP[T.CUP][T.FW]}{3600} \tag{6.1}$$

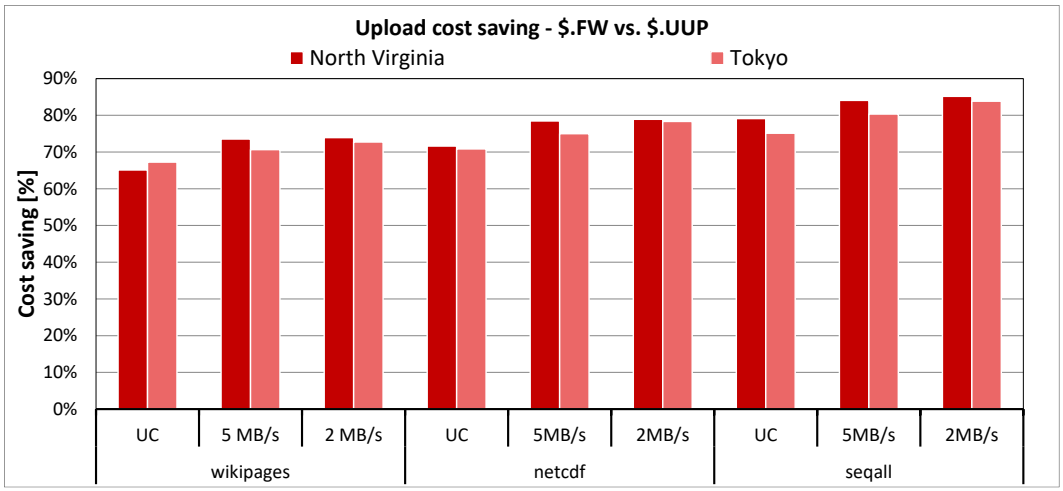
$$\$.UDW = \frac{Util_{FEE} \cdot T.UDW}{3600} + \frac{Dout_{FEE} \cdot US}{1024} \tag{6.2}$$

$$\$.CDW[\$.FW] = \frac{Util_{FEE} \cdot T.CDW[T.FW]}{3600} + \frac{Dout_{FEE} \cdot \frac{US}{CR}}{1024} \tag{6.3}$$

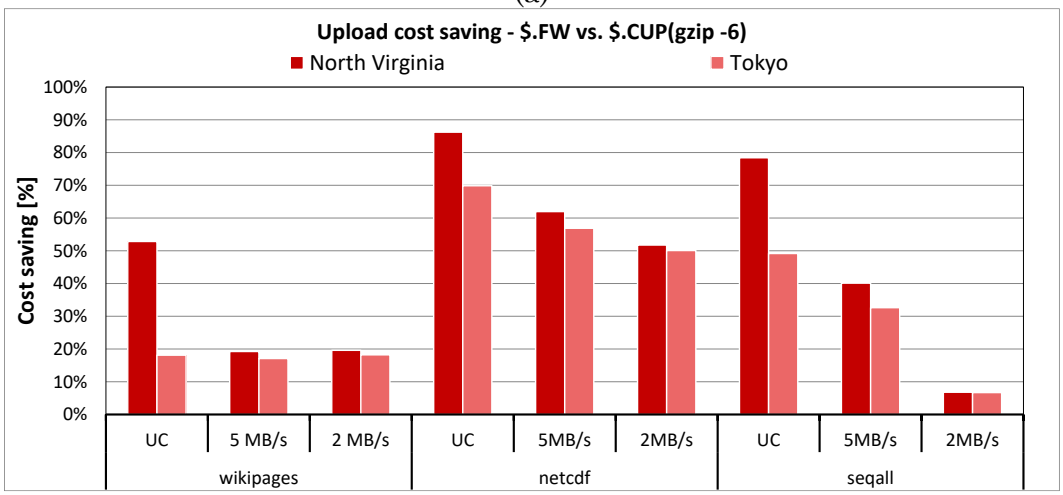
The overall cost savings achieved by the framework relative to the uncompressed transfers are calculated as the percentage of the cost saved from the total cost of the uncompressed transfers, *($.UUP-$.FW)/$.UUP [($.UDW-$.FW)/$.UDW]*. The overall cost savings achieved by the framework relative to the default compressed transfers are calculated as the percentage of the cost saved from the total cost of the default compressed transfers, *($.CUP(gzip -6)-$.FW)/$.CUP(gzip -6) [($.CDW(gzip -6)-$.FW)/$.CDW(gzip -6)]*.

6.3.2.1 Cost Savings for Uploads

($.UUP-$.FW)/$.UUP. Figure 6.13 (a) shows the overall cost savings achieved by the framework relative to the uncompressed file uploads. Each bar represents the overall cost savings of transferring the subset of 20 files from each dataset to the North Virginia and Tokyo cloud instances. The dark red bars represent cost savings for uploads to the North Virginia instance, and they range from 65.1% to 79.1% with the uncapped network, and up to 73.9% and 85.2% for the 2 MB/s network. The light red bars represent cost savings for uploads to the Tokyo instance and are similar to those observed for the North Virginia instance.



(a)



(b)

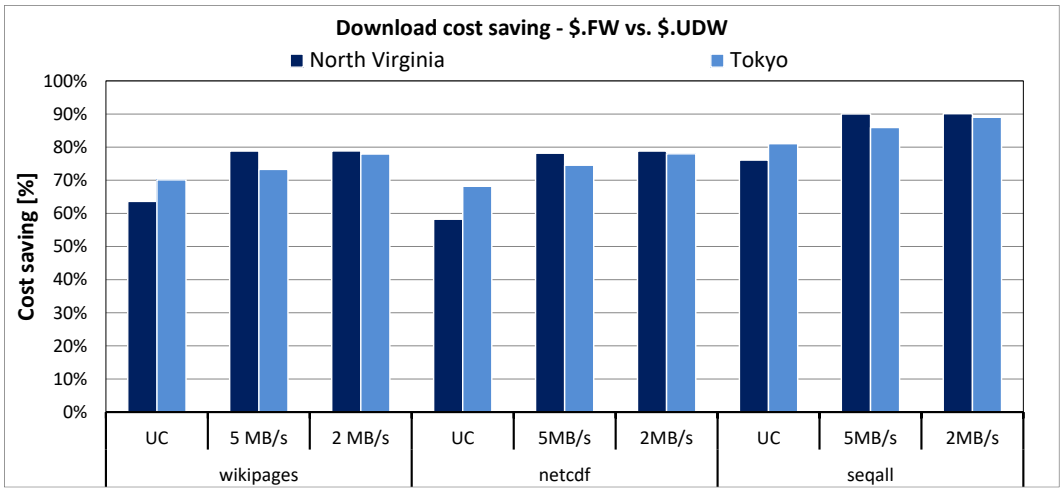
Figure 6.13 Upload cost savings for North Virginia and Tokyo transfers: $.FW vs. $.UUP (a) and $.FW vs. $.CUP(gzip -6) (b)

($.CUP(gzip -6)-$.FW)/$.CUP(gzip -6). Figure 6.13 (b) shows the overall cost savings achieved by the framework relative to the default compressed file uploads. The dark red bars represent cost savings for uploads to the North Virginia instance, and they range from 52.8% to 86.2% with the uncapped network, from 40.1% to 62.0% with the 5 MB/s network, and from 6.8% to 51.8% with the 2 MB/s network. The light red bars represent cost savings for uploads to the Tokyo instance, and they

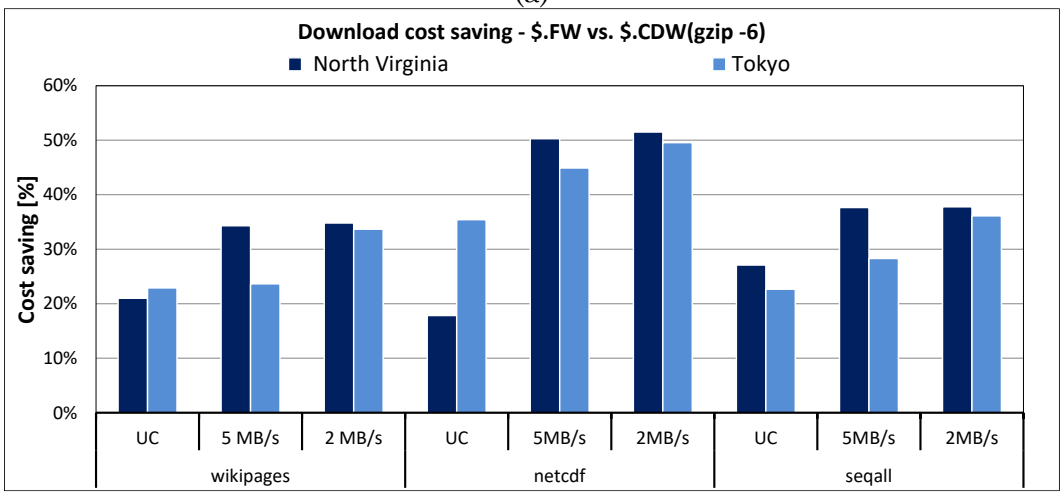
range from 18.10% to 69.84% with the uncapped network, from 17.09% to 56.87% with the 5 MB/s network, and from 6.69% to 50.06% with the 2 MB/s network. The cost saving depends on the input file (e.g., they are higher for the *netcdf* dataset), and the network throughput (highest for the North Virginia cloud with the uncapped network).

6.3.2.2 Cost Saving for Downloads

($.UDW-$.FW)/$.UDW. Figure 6.13 (a) shows the overall cost savings achieved by the framework relative to the uncompressed file downloads. Each bar represents the overall cost savings of transferring the subset of 20 files from each dataset from the North Virginia and Tokyo cloud instances. The dark blue bars represent cost savings for downloads from the North Virginia instance, and they range from 58.2% to 78.2% with the uncapped network, and from 78% to 90.1% for the capped network at 5MB/s and 2 MB/s. The light blue bars represent cost savings for downloads from the Tokyo instance. The cost savings are close to the cost savings achieved with downloads from the North Virginia instance.



(a)



(b)

Figure 6.14 Download cost saving for North Virginia and Tokyo transfers: $.FW vs. $.UDW (a) and $.FW vs. $.CDW(gzip -6) (b)

($.CDW(gzip -6)-$.FW)/$.CDW(gzip -6). Figure 6.13 (b) shows cost savings achieved by the framework relative to the default compressed file. The dark blue bars represent cost savings for downloads from the North Virginia instance, and they range from 17.8% to 27.1% with the uncapped network, from 34.3% to 50.3% with the 5 MB/s network, and from 34.8% to 51.5% with the 2 MB/s network. The light blue bars represent cost savings for downloads from the Tokyo instance and are

slightly less than those observed for the North Virginia instance. The cost saving depends on the input file (e.g., higher for *netcdf*), and the network throughput (highest on the 2 MB/s).

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

This dissertation describes the design, an implementation, and an experimental evaluation of the framework for optimal data transfers between edge devices and the cloud using compression utilities. The framework seamlessly selects a file transfer mode that maximizes either effective throughput or energy-efficiency – the file can be transferred uncompressed or using any compression (utility, level) pair from a set of available options. The framework relies on the following to select an effective transfer mode: (i) parameters describing an input file; (ii) parameters describing the network connection; (iii) analytical models describing effective throughputs and energy-efficiency; and (iv) parameters describing the device that initiates the transfer. Software agents running on edge devices and the cloud utilize analytical models, history-based tables, relevant parameters for a given transfer, and optimization mode to select an effective transfer mode. The framework implementations are deployed on Android mobile devices and Linux-based workstations.

The effectiveness of the proposed framework is experimentally evaluated on mobile devices and workstations while performing upload and download file transfers for a range of input files and network types and conditions. The performance of the framework is compared to the performance of uncompressed transfers and the default compressed file transfers that use *gzip* with -6 compression level. For the mobile device used in this study, the framework with the throughput optimization

mode improves effective upload throughputs from 1.50 (on the 0.5 MB/s WLAN) to 2.54 (on the 5 MB/s WLAN) times relative to the uncompressed uploads and from 1.01 to 2.89 times relative to the default compressed uploads. It improves effective download throughputs from 1.78 to 2.52 times relative to the uncompressed downloads and up to 1.2 times relative to the default compressed downloads. The framework with the energy efficiency optimization mode improves the effective upload energy efficiencies from 1.09 to 1.16 times relative to the uncompressed uploads and from 1.09 to 2.13 times relative to the default compressed uploads. It improves the effective download energy efficiencies from 1.45 to 1.24 times relative to the uncompressed downloads and from 1.05 to 1.17 times relative to the default compressed downloads. For the workstation with transfers to and from the cloud instance in North Virginia, the framework with the throughput optimization mode improves the effective upload throughputs from 3.51 (on the uncapped network) to 4.76 (on the 2 MB/s network) times relative to the uncompressed uploads and from 4.36 to 1.48 times relative to the default compressed uploads. It improves the effective download throughputs from 2.30 to 5.64 times relative to the uncompressed downloads and from 1.29 to 1.75 times relative to the default compressed downloads. Similar improvements to throughputs are achieved with transfers to and from the cloud instance in Tokyo. In addition to throughput improvements, the framework provides a reduction of costs associated with file transfers to and from the cloud, which includes the costs of using cloud resources and the costs of transferring files out of the cloud. With transfers to and from the cloud instance in North Virginia, the framework reduces the total costs by 60% (uncapped) to 90% (2 MB/s) relative to the uncompressed uploads and by 70% to 30% relative to the default compressed uploads. It

reduces the total costs by 60% to 90% relative to the uncompressed downloads and by 20% to 50% relative to the default compressed downloads.

The use of this framework is not limited to six lossless utilities introduced in Section 2.1, and can be expanded to any number of utilities, as long as prediction data for characterization of new compression (utility, level) pairs (e.g., local throughput, energy efficiency, and compression ratio) is made available to the analytical models described in Chapter 4. For example, the framework can be expanded to contain other compression utilities such as *LZ4* [86], *lbzip2* [87], Google's *Snappy* [88] and *zopfli* [89]. *LZ4* and *Snappy* could offer faster decompressions, expected to be useful in networks with higher network throughputs, while *lbzip2* could offer better performance relative to utilities such *xz* and *bzip2*. This flexibility will allow for the use of the framework on any future platforms and compression utilities. Introducing new utilities, especially file, application, or platform-specific, will increase the overall effectiveness of the framework on speedup in throughput, energy efficiency, and increase in the overall cost savings.

The future improvements to the framework design should optimize currently required pre-processing steps of generating system and device specific history-based prediction data for selected compression (utility, level) pairs. These steps can be simplified with a deeper analysis of compression utilities or with the use of machine learning techniques. The analytical models for estimating energy efficiency of uncompressed and compressed data transfers should be improved by relying on the performance counters. Finally, the SQL query execution times, a limiting factor for uploads from the mobile devices, and for transfers of smaller files in general, should be optimized through GPU and FPGA acceleration on edge devices and the cloud.

Both mobile devices and workstations can exploit hardware accelerators by utilizing OpenCL or CUDA.

REFERENCES

[1] Gartner, Inc., "Gartner Says Smartphone Sales Surpassed One Billion Units in 2014," 2014. [Online]. Available: http://www.gartner.com/newsroom/id/2996817. [Accessed: 16-Feb-2015].

[2] Gartner, Inc., "Gartner Says Tablet Sales Continue to Be Slow in 2015," 2014. [Online]. Available: http://www.gartner.com/newsroom/id/2954317. [Accessed: 26-Jan-2015].

[3] Gartner, Inc., "Market Share Analysis: Mobile Phones, Worldwide, 4Q13 and 2013," 2014. [Online]. Available: http://www.gartner.com/newsroom/id/2665715. [Accessed: 16-Jan-2015].

[4] Gartner, Inc., "Forecast: PCs, Ultramobiles, and Mobile Phones, Worldwide, 2011-2018, 1Q14 Update," 2014. [Online]. Available: http://www.gartner.com/newsroom/id/2692318. [Accessed: 26-Jan-2015].

[5] CISCO, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update 2014–2019 White Paper." 03-Feb-2015.

[6] "GSMA Mobile Economy 2015." GSMA, 2015.

[7] "Folding@home," *Folding@home*. [Online]. Available: http://folding.stanford.edu. [Accessed: 09-Dec-2015].

[8] "SAT@home." [Online]. Available: http://sat.isa.ru/pdsat/. [Accessed: 09-Dec-2015].

[9] B. R. Badrinath, A. Acharya, and T. Imielinski, "Designing distributed algorithms for mobile computing networks," *Computer Communications*, vol. 19, no. 4, pp. 309–320, Apr. 1996.

[10] V. Agababov *et al.*, "Flywheel: Google's Data Compression Proxy for the Mobile Web," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, Berkeley, CA, USA, 2015, pp. 367–380.

[11] Google, "Data Server - Google Chrome," 2014. [Online]. Available: https://developer.chrome.com/multidevice/data-compression. [Accessed: 30-Oct-2015].

[12] Amazon, "What Is Amazon Silk? - Amazon Silk," 2015. [Online]. Available: http://docs.aws.amazon.com/silk/latest/developerguide/. [Accessed: 06-Dec-2015].

[13] Onavo, "Onavo," *Onavo*, 2015. [Online]. Available: http://www.onavo.com. [Accessed: 01-Dec-2015].

[14] Snappli, "Snappli," 2014. [Online]. Available: http://snappli.com/. [Accessed: 01-Dec-2015].

[15] zlib, "zlib Home Site," 2015. [Online]. Available: http://www.zlib.net/. [Accessed: 16-Dec-2015].

[16] Google, "About Attachment Manager," 2014. [Online]. Available: http://www.google.com/support/enterprise/static/postini/docs/admin/en/admin_msd/attach_overview.html. [Accessed: 31-Oct-2015].

[17] A. Dzhagaryan, A. Milenkovic, and M. Burtscher, "Energy efficiency of lossless data compression on a mobile device: An experimental evaluation," in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, Austin, TX, 2013, pp. 126–127.

[18] A. Milenkovic, A. Dzhagaryan, and M. Burtscher, "Performance and Energy Consumption of Lossless Compression/Decompression Utilities on Mobile Computing Platforms," in *2013 IEEE 21st International Symposium on Modeling,*

Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), San Francisco, CA, 2013, pp. 254–263.

[19] A. Dzhagaryan, A. Milenkovic, and M. Burtscher, "Quantifying Benefits of Lossless Compression Utilities on Modern Smartphones," in *2015 24th International Conference on Computer Communication and Networks (ICCCN)*, Las Vegas, NV, 2015, pp. 1–9.

[20] A. Dzhagaryan and A. Milenkovic, "On Effectiveness of Lossless Compression in Transferring mHealth Data Files," in *2015 IEEE 17th International Conference on e-Health Networking, Applications and Services (Healthcom)*, Boston, MA, 2015.

[21] A. Dzhagaryan and A. Milenković, "Analytical Models for Evaluating Effectiveness of Compressed File Transfers in Mobile Computing," in *Proceedings of the 6th International Joint Conference on Pervasive and Embedded Computing and Communication Systems - Volume 1: PEC,* 2016, pp. 40–51.

[22] A. Dzhagaryan and A. Milenkovic, "Models for Evaluating Effective Throughputs for File Transfers in Mobile Computing," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, Waikoloa, HI, 2016.

[23] K. Barr and K. Asanović, "Energy aware lossless data compression," in *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services (MobiSys'03)*, 2003, pp. 231–244.

[24] K. C. Barr and K. Asanović, "Energy-aware lossless data compression," *ACM Transactions on Computer Systems*, vol. 24, no. 3, pp. 250–291, Aug. 2006.

[25] Google, “Google Play,” 2015. [Online]. Available: https://play.google.com. [Accessed: 13-Dec-2015].

[26] “Dropbox.” [Online]. Available: https://www.dropbox.com/. [Accessed: 13-Dec-2015].

[27] Google, “Google Drive - Cloud Storage & File Backup for Photos, Docs & More,” 2015. [Online]. Available: https://www.google.com/drive/. [Accessed: 13-Dec-2015].

[28] “NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy,” *NGINX*. [Online]. Available: https://www.nginx.com/. [Accessed: 02-Oct-2015].

[29] “Welcome to The Apache Software Foundation!” [Online]. Available: http://www.apache.org/. [Accessed: 02-Oct-2015].

[30] “The gzip home page.” [Online]. Available: http://www.gzip.org/. [Accessed: 25-May-2012].

[31] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” *IEEE Transaction on Information Theory*, vol. 23, pp. 337–343, 1977.

[32] M. Oberhumer, “lzop file compressor (oberhumer.com OpenSource).” [Online]. Available: http://www.lzop.org/. [Accessed: 25-May-2012].

[33] “bzip2 : Home.” [Online]. Available: http://www.bzip.org/. [Accessed: 25-Sep-2015].

[34] M. Burrows and D. J. Wheeler, “A Block-sorting Lossless Data Compression Algorithm,” Digital SRC, Report 124, May 1994.

[35] “XZ Utils.” [Online]. Available: http://tukaani.org/xz/. [Accessed: 25-May-2012].

[36] I. Pavlov, “7-Zip.” [Online]. Available: http://www.7-zip.org/. [Accessed: 25-May-2012].

[37] “pigz - Parallel gzip.” [Online]. Available: http://zlib.net/pigz/. [Accessed: 25-May-2012].

[38] “pigz - Parallel gzip,” 2015. [Online]. Available: http://zlib.net/pigz/. [Accessed: 25-Aug-2015].

[39] J. Gilchrist, “Parallel BZIP2 (PBZIP2).” [Online]. Available: http://compression.ca/pbzip2/. [Accessed: 25-May-2015].

[40] “F-Droid | Free and Open Source Android App Repository.” .

[41] “ownCloud.org.” [Online]. Available: https://owncloud.org/. [Accessed: 01-Oct-2015].

[42] “Nextcloud.” [Online]. Available: https://nextcloud.com/. [Accessed: 25-Sep-2016].

[43] “Nexus.” [Online]. Available: http://www.google.com/nexus/. [Accessed: 26-Nov-2015].

[44] OnePlus, “OnePlus One,” 2015. [Online]. Available: https://oneplus.net/one. [Accessed: 12-Jul-2015].

[45] Qualcomm, “Snapdragon™ Mobile Processors - Qualcomm Developer Network,” 2014. [Online]. Available: https://developer.qualcomm.com/discover/chipsets-and-modems/snapdragon. [Accessed: 20-Jun-2014].

[46] Qualcomm, “Adreno,” *Qualcomm Developer Network*, 2015. [Online]. Available: https://developer.qualcomm.com/software/adreno-gpu-sdk/gpu. [Accessed: 14-Jun-2015].

[47] CyanogenMod, “CyanogenMod | Android Community Operating System,” 2014. [Online]. Available: http://www.cyanogenmod.org/. [Accessed: 14-Jun-2014].

[48] “Intel® Turbo Boost Technology 2.0,” *Intel*. [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html. [Accessed: 20-Aug-2016].

[49] J. Treibig, G. Hager, and G. Wellein, “LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments,” in *2010 39th International Conference on Parallel Processing Workshops (ICPPW)*, 2010, pp. 207–216.

[50] J. Treibig, G. Hager, and G. Wellein, “LIKWID: Lightweight Performance Tools,” *arXiv:1104.4874 [cs]*, pp. 207–216, Sep. 2010.

[51] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, “Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge,” *IEEE Micro*, vol. 32, no. 2, pp. 20–27, Mar. 2012.

[52] NI, “NI PXIe-4154 - National Instruments,” 2014. [Online]. Available: http://sine.ni.com/nips/cds/view/p/lang/en/nid/209090. [Accessed: 20-Jun-2014].

[53] NI, “NI PXIe-1073 - National Instruments,” 2014. [Online]. Available: http://sine.ni.com/nips/cds/view/p/lang/en/nid/207401. [Accessed: 20-Jun-2014].

[54] “Android Debug Bridge | Android Developers,” 2015. [Online]. Available: http://developer.android.com/tools/help/adb.html. [Accessed: 14-Jun-2015].

[55] Google, “Log | Android Developers,” 2014. [Online]. Available: http://developer.android.com/reference/android/util/Log.html. [Accessed: 03-Aug-2014].

[56] A. Dzhagaryan, A. Milenković, M. Milosevic, and E. Jovanov, “An environment for automated measurement of energy consumed by mobile and embedded computing devices,” *Measurement*, vol. 94, pp. 103–118, Dec. 2016.

[57] A. Dzhagaryan, A. Milenkovic, M. Milosevic, and E. Jovanov, "An Environment for Automated Measuring of Energy Consumed by Android Mobile Devices," in *6th International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS)*, Lisbon, Portugal, 2016.

[58] "PAPI." [Online]. Available: http://icl.cs.utk.edu/papi/index.html. [Accessed: 20-Dec-2013].

[59] V. M. Weaver *et al.*, "Measuring Energy and Power with PAPI," in *2012 41st International Conference on Parallel Processing Workshops (ICPPW)*, 2012, pp. 262–268.

[60] "Perf Wiki." [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page. [Accessed: 07-Dec-2013].

[61] "MAPS.ME (MapsWithMe), detailed offline maps of the World for iPhone, iPad, iPod, Android, Amazon Kindle Fire and BlackBerry," 2015. [Online]. Available: http://maps.me. [Accessed: 01-Oct-2015].

[62] OsmAnd, "OsmAnd - Offline Mobile Maps and Navigation," 2015. [Online]. Available: http://osmand.net/. [Accessed: 01-Oct-2015].

[63] "Home - UniGene - NCBI." [Online]. Available: http://www.ncbi.nlm.nih.gov/unigene. [Accessed: 29-May-2016].

[64] Amante, C. and B. W. Eakins, "ETOPO1 1 Arc-Minute Global Relief Model: Procedures, Data Sources and Analysis." National Geophysical Data Center, NOAA, 2009.

[65] "enwiki dump." [Online]. Available: https://dumps.wikimedia.org/enwiki/20160407/. [Accessed: 30-May-2016].

[66] "IEEE Xplore - Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?" [Online]. Available:

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5445167. [Accessed: 02-Nov-2012].

[67] N. K. Nithi and A. J. de Lind van Wijngaarden, "Smart power management for mobile handsets," *Bell Lab. Tech. J.*, vol. 15, no. 4, pp. 149–168, Mar. 2011.

[68] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture, 2009. MICRO-42*, 2009, pp. 168–178.

[69] A. Rice and S. Hay, "Decomposing power measurements for mobile devices," in *2010 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2010, pp. 70–78.

[70] A. Rice and S. Hay, "Measuring mobile phone energy consumption for 802.11 wireless networking," *Pervasive and Mobile Computing*, vol. 6, no. 6, pp. 593–606, Dec. 2010.

[71] M. Milosevic, A. Dzhagaryan, E. Jovanov, and A. Milenković, "An Environment for Automated Power Measurements on Mobile Computing Platforms," in *Proceedings of the 51st ACM Southeast Conference*, New York, NY, USA, 2013, p. 6.

[72] A. Carroll and G. Heiser, "An Analysis of Power Consumption in a Smartphone," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, Berkeley, CA, USA, 2010, pp. 21–21.

[73] W. L. Bircher and L. K. John, "Complete System Power Estimation Using Processor Performance Events," *IEEE Transactions on Computers*, vol. 61, no. 4, pp. 563–577, Apr. 2012.

[74] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proceedings of the sixth conference on Computer systems*, New York, NY, USA, 2011, pp. 153–168.

[75] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: fine grained energy accounting on smartphones with Eprof," in *Proceedings of the 7th ACM european conference on Computer Systems*, New York, NY, USA, 2012, pp. 29–42.

[76] T. Li and L. K. John, "Run-time modeling and estimation of operating system power consumption," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 160–171, Jun. 2003.

[77] B. Nicolae, "High Throughput Data-Compression for Cloud Storage," in *Data Management in Grid and Peer-to-Peer Systems*, A. Hameurlain, F. Morvan, and A. M. Tjoa, Eds. Springer Berlin Heidelberg, 2010, pp. 1–12.

[78] D. Harnik, R. Kat, O. Margalit, D. Sotnikov, and A. Traeger, "To Zip or Not to Zip: Effective Resource Usage for Real-time Compression," in *Proceedings of the 11th USENIX Conference on File and Storage Technologies*, Berkeley, CA, USA, 2013, pp. 229–242.

[79] S. Raskhodnikova, D. Ron, R. Rubinfeld, and A. Smith, "Sublinear Algorithms for Approximating String Compressibility," *Algorithmica*, vol. 65, no. 3, pp. 685–709, Feb. 2012.

[80] J. K. Bonfield and M. V. Mahoney, "Compression of FASTQ and SAM Format Sequencing Data," *PLoS ONE*, vol. 8, no. 3, pp. 1–10, 2013.

[81] A. Guerra, J. Lotero, and S. Isaza, “Performance comparison of sequential and parallel compression applications for DNA raw data,” *J Supercomput*, pp. 1–22, Jun. 2016.

[82] A. Carroll and G. Heiser, “An analysis of power consumption in a smartphone,” in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, Berkeley, CA, USA, 2010, pp. 21–21.

[83] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, “RAPL: Memory power estimation and capping,” in *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, 2010, pp. 189–194.

[84] “Reading RAPL energy measurements from Linux.” [Online]. Available: http://web.eece.maine.edu/~vweaver/projects/rapl/. [Accessed: 09-Oct-2016].

[85] B. Klika and C. Jordan, “High-intensity circuit training using body weight: Maximum results with minimal investment,” *ACSM’s Health & Fitness Journal*, vol. 17, no. 3, pp. 8–13, 2013.

[86] “LZ4 - Extremely fast compression,” 03-Sep-2016. [Online]. Available: http://cyan4973.github.io/lz4/. [Accessed: 03-Sep-2016].

[87] “lbzip2 - parallel bzip2 compression utility,” 03-Sep-2016. [Online]. Available: http://lbzip2.org/. [Accessed: 03-Sep-2016].

[88] “Snappy by google.” [Online]. Available: https://google.github.io/snappy/. [Accessed: 03-Sep-2016].

[89] “google/zopfli,” *GitHub*. [Online]. Available: https://github.com/google/zopfli. [Accessed: 03-Sep-2016].