Quantifying Benefits of Lossless Compression Utilities on Modern Smartphones

Armen Dzhagaryan, Aleksandar Milenković
Electrical and Computer Engineering
The University of Alabama in Huntsville
Huntsville, AL

Martin Burtscher
Computer Science
Texas State University, San Marcos
San Marcos, TX

Abstract — The data traffic originating on mobile computing devices has been growing exponentially over the last several years. Lossless data compression and decompression can be essential in increasing communication throughput, reducing communication latency, achieving energy-efficient communication, and making effective use of available storage. This paper experimentally evaluates several compression utilities and configurations on a modern smartphone. We characterize each utility in terms of its compression ratio, compression and decompression throughput, and energy efficiency for representative use cases. We find a wide variety of energy costs associated with data compression and decompression and provide practical guidelines for selecting the most energy efficient configurations for each use case. For data transfers over WLAN, the best configurations provide a 2.1-fold and 2.7-fold improvement in energy efficiency for compressed uploads and downloads, respectively, when compared to uncompressed data transfers. For data transfers over a mobile broadband network, the best configurations provide a 2.7-fold and 3-fold improvement in energy efficiency for compressed uploads and downloads, respectively.[1]

Index Terms — Mobile computing, Measurement techniques, Data compression, Energy-aware systems

I. Introduction

Mobile computing devices such as smartphones, tablet computers, and e-readers have become the dominant platforms for consuming digital information. According to estimates for 2013 [1], [2], [3], vendors shipped 968 million smartphones (up 42.3% from the prior year) and 195 million tablets (up 68%), whereas the number of notebooks and desktop computers shipped was 296.1 million. Annual sales of smartphones for the first time exceeded annual sales of feature phones [1], totaling 1,807 million mobile phones shipped in 2013 [3]. Global mobile data traffic continues to grow exponentially. A report from Cisco states that the global mobile data traffic grew 1.81-fold in 2013, reaching 1.5 exabytes per month, which is over 18 times greater than the total Internet traffic in the year 2000 [4]. It is forecast that the global mobile data traffic will grow 11-fold from 2013 to 2018, reaching 15.9 exabytes per month.

Data compression is critical in mobile data communication. It can help improve operating time, lower communication latencies, 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 contrast, 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 article.

Lossless data compression is currently being used to reduce the required bandwidth during file downloads and to speed up webpage loads in browsers. Google and Amazon [5], as well as the mobile applications Onavo Extend and Snappli [6], [7], use proxy servers to provide HTTP compression for all pages during web browsing. For file downloads, several Google services [8], such as Gmail and Drive, provide zip compression of files and attachments. Similarly, most software repositories (including mobile repositories such as Google Play and Apple App Store) are using zip or zip-derived containers for data distribution. Several Linux distributions are also using gzip, bzip2, and xz for their software repositories.

The choice of compression algorithm, the compression level, and the quality of the implementation affect the performance and energy consumption. Whereas the energy consumed for compression and decompression tasks is not critical on desktop PCs and workstations, it can be a decisive factor in battery-powered mobile devices. Achieving a higher compression ratio requires more computation and therefore energy, but better compression reduces the number of bytes, thus saving energy when transmitting the data. Hence, we believe it is important to take a close look at both the performance and the energy efficiency of lossless compression algorithms on state-of-the-art smartphones. *In particular, we want to determine whether compression is useful for reducing the latency and energy consumption of data transfers in mobile environments, what common compression algorithms should be used, which configurations result in the best performance and energy efficiency, and whether parallel implementations can help. More-*

[1] This material is based upon work supported in part by the National Science Foundation under Grants No. 1141022, 1205439, 1217231, 1217470, 1406304, and 1438963. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

over, we want to develop a framework for determining if compression should be used and what configuration will provide the best results.

In this article, we perform a comparative, measurement-based study of the most recent versions of several popular compression utilities, including *gzip*, *lzop*, *bzip2*, *xz*, *pigz* (a parallel implementation of *gzip*) and *pbzip2* (a parallel implementation of *bzip2*) on Google's Nexus 4 smartphone. For each utility, we analyze the effectiveness of all supported compression levels. We examine several performance metrics, including the compression ratio and the compression and decompression throughput. Using our experimental setup for energy measurements, we study the amount of energy consumed by compression and decompression tasks and report the energy efficiency.

We evaluate the compression utilities in three typical usage scenarios. *LOCAL* involves compression and decompression tasks performed locally on the smartphone. *WLAN* and *CELL* involve compression tasks that stream data to and from a remote server over a secure communication channel. *WLAN* uses a wireless LAN interface and *CELL* uses a mobile broadband network. *We also devise an analytical model for estimating the effectiveness of individual utilities in the WLAN and CELL scenarios. The model relies on parameters from the LOCAL experiments and parameters characterizing the network interface.*

Our main findings are as follows.

- The throughput and energy efficiency of compression utilities varies widely across different utilities and compression levels, often spanning an order of magnitude. We identify combinations of the utilities and compression levels that result in the best throughput and energy efficiency for typical use scenarios.
- In the LOCAL experiments, *pigz* with compression level -1 and *lzop* -1 to -6 achieve the best compression throughput and energy efficiency. *lzop* achieves the best decompression throughput and efficiency.
- In the *WLAN* experiments, compressed uploads with *gzip* with a low compression level perform best, providing over twice more energy efficient transfers compared to uncompressed uploads. For downloads, *pigz* with compression levels -6 to -9 achieves the best decompression throughput and energy efficiency, providing ~2.7 times more energy efficient transfers compared to uncompressed downloads.
- In the CELL experiments, compressed uploads with *bzip2* perform best, with up to ~2.7 times more energy efficient transfers compared to uncompressed uploads. For decompression tasks, *xz* with the highest compression levels achieves the best decompression throughput and energy efficiency, providing up to 3 times more energy efficient transfers compared to uncompressed downloads.

The importance of lossless compression on consumer devices has been widely recognized [9]. This article complements our earlier study on Linux-based mobile development platforms with limited connectivity [10], [11] and other studies that were conducted about a decade ago [12], [13]. In contrast to these prior studies, here we consider the most recent compression utilities including some with parallel implementations, our setup supports more accurate energy measurements, and we use a state-of-the-art smartphone with four processor cores and wireless and cellular communication interfaces. In addition, we study the performance and energy efficiency for all supported compression levels in three typical use scenarios with representative datasets.

The rest of this paper is organized as follows. Section II presents the experimental setup, including the smartphone (II.A), the measurement setup (II.B), the compression utilities (II.C), and the dataset (II.D). Section III explains the measured and derived metrics used (III.A) as well as the experiments (III.B). Section IV discusses the results. Section V summarizes our findings and draws conclusions.

II. Experimental Setup

A. Smartphone

We use Google's Nexus 4 smartphone [14] as the target platform. The Nexus 4 is powered by a Qualcomm Snapdragon S4 Pro (APQ8064) system-on-a-chip that features a quad core ARM processor running at up to 1.512 GHz clock frequency. The smartphone has 2 GBytes of LPDDR2 RAM memory and 16 GBytes of built-in internal storage. It supports a range of connectivity options including WLAN 802.11n, Bluetooth 4.0, and several cellular network protocols.

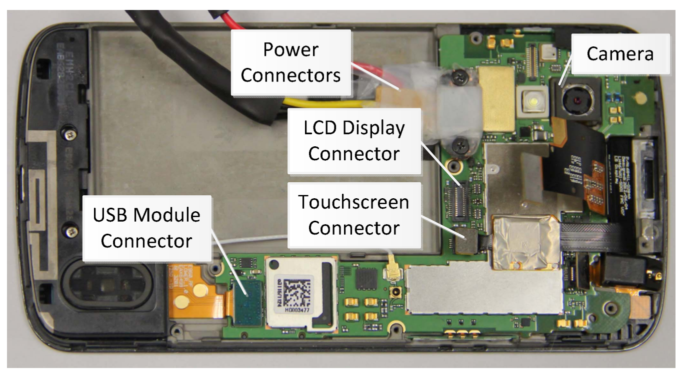


Fig. 1. Nexus 4 prepared for energy measurements

To prepare the smartphone for energy profiling, its underlying plastic shield was removed to reveal connections on its motherboard and daughterboards as shown in Fig. 1. We replaced the smartphone's battery with power connectors coming from a battery simulator. During measurements, connectors to the smartphone components such as the LCD display, touchscreen, USB and others can be removed to reduce the impact of these components on the consumed energy.

The smartphone's Android (Jelly Bean) operating system is upgraded to (a) support common compression/decompression utilities not readily supported on Android, and (b) to automate performance and energy measurements. We flashed the smartphone with CyanogenMod version 10.2 [15], an open source operating system for smartphones and tablet computers based on official releases of Android that include third-party code. The included compression utilities are *gzip*, *bzip2*, *lzop*, and *xz*. The parallel versions of *gzip* and *bzip2*, *pigz* and *pbzip2*, respectively, were compiled on the smartphone.

B. *Measuring setup*

Fig. 2 shows a block diagram of our setup for measuring the energy consumed on the smartphone. The main components of the setup are an NI PXIe-4154 battery simulator, the smartphone, and a workstation. The battery simulator, a specialized programmable power supply, resides inside an NI PXIe-1073 chassis that is connected to an MXI-Express Interface card inside the workstation. The battery simulator is used (a) to power the smartphone through probes on channel 0 by providing 4.1 volts, thus bypassing the actual smartphone battery, and (b) to measure the current drawn by the smartphone while running applications. The battery simulator samples the current with a configurable sampling rate of up to 200,000 samples per second and a sensitivity of 1 μA.

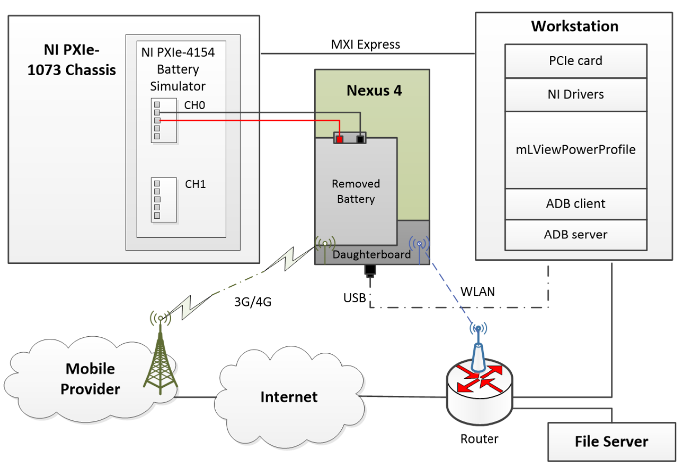


Fig. 2. Block diagram of the hardware setup for energy profiling

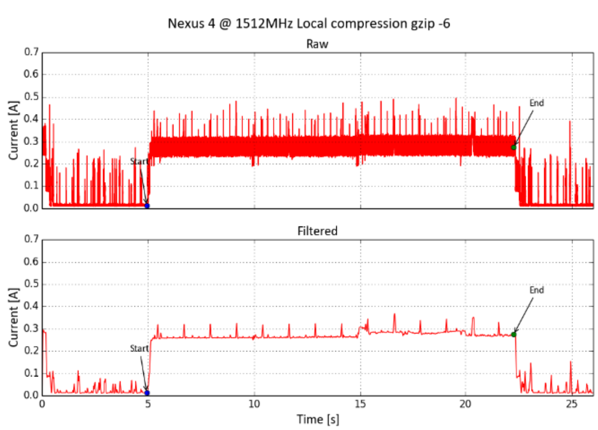


Fig. 3. Current drawn by the Nexus 4 when executing the gzip utility

The workstation runs our custom program called *mLViewPowerProfile*. This program interfaces (a) the smartphone to manage activities and applications on the smartphone that are being profiled and (b) the battery simulator to collect the current samples. The communication with the smartphone is carried out over the Android Debug Bridge (*adb*) [16]. *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 workstation; and a daemon, which runs on the smartphone. *adb* can connect over a USB link or over a WLAN 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. The execution of a compression/decompression 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 compression/decompression task is wrapped by commands that take time stamps corresponding to the moments when the task is launched and finishes. 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.

Fig. 3 shows the measured current drawn by the Nexus 4 during execution of a script that compresses an input file using *gzip* -6 and outputs the result to /dev/null. 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 by a human 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 (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.

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

$$ET.C(0) = ET.C - I_{idle} \cdot V_{BS} \cdot T.C \tag{2}$$

In addition to *ET.C*, we also calculate the energy overhead 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).

C. *Compression Utilities*

TABLE I lists the six lossless compression utilities we have studied along with the supported range of compression levels. We chose the relatively fast *gzip* utility and the slower but better compressing *bzip2* utility because of their widespread use. *lzop* is included because of its high speed. *xz* is gaining ground and is known for its high compression ratio, relatively slow compression, and fast decompression. As many modern smartphones include multicore CPUs, we also included *pigz* and *pbzip2*, which are parallel versions of *gzip* and *bzip2*, respectively. All of these utilities operate at byte 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. *gzip* [17] implements the deflate algorithm, which is a variant of the LZ77 algorithm [18]. *lzop* [19] uses a block-based compression algorithm that favors speed over compression ratio and requires very little memory for decompression. The implementation ported on our test device supports only compression levels -1 to -6. *bzip2* [20] implements a variant of the block-sorting algorithm described by Burrows and Wheeler (BWT) [21]. *xz* [22] is based on the Lempel-Ziv-Markov chain compression algorithm (LZMA) developed for 7-Zip [23]. *pigz* [24] 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. *pbzip2* [25] is a multithreaded version of *bzip2* that is also based on pthreads. It works by compressing multiple blocks of data simultaneously. The resulting blocks are then concatenated to form the final compressed file, which is compatible with *bzip2*. Decompression is also parallelized.

TABLE I
COMPRESSION UTILITIES

Utility	Compression levels	Version	Notes
gzip	1 – 9 (6)	1.6	DEFLATE (Ziv-Lempel, Huffman)
lzop	1 – 9 (3)	1.03	LZO (Lempel-Ziv-Oberhumer)
bzip2	1 – 9 (9)	1.0.6	RLE+BWT+MTF+RLE+Huffman
xz	0 – 9 (6)	5.1.0a	LZMA2
pigz	1 – 9 (6)	2.3	Parallel implementation of gzip
pbzip2	1 – 9 (9)	1.1.6	Parallel implementation of bzip2

TABLE II
DATASET

i	Type	Raw size [KB]	Notes
1	text (txt)	15,711.6	Project Gutenberg Works of Mark Twain
2	exec (so)	12,452.5	Open source web content engine library
3	image (bmp)	16,777.2	An image of Earth from space
4	table (csv)	9,988.9	Activity data from a health monitor
5	code (tar)	11,233.2	Perl 5.8.5 source code

D. Datasets

In selecting the data to evaluate the effectiveness of the compression utilities, we compiled a set of diverse input files that are representative of mobile computing environments. The input file formats include text, an executable, an image, a file with comma-separated values from a wearable health monitor, and source code. TABLE II gives the input files, their types, size in bytes, and a description. The files are merged into a single archive file (tar) that is used as an input for the compression utilities.

III. METRICS AND EXPERIMENTS

A. Metrics

TABLE III summarizes the metrics used as well as their definitions.

Compression ratio. We use the compression ratio (CR=US/CS) to evaluate the compression effectiveness of an individual utility and its levels of compression.

Performance. To evaluate the performance of individual compression utilities and compression levels, we measure the time to compress the raw input file (T.C) and the time to decompress (T.D) a compressed file generated by that utility with the selected compression level. Each compression and decompression task is repeated three times and the average time is calculated. Instead of reporting the execution times directly, we report the compression and decompression throughput (Th.C and Th.D) expressed in megabytes per second (Th.C=US/T.C and Th.D=US/T.D).

TABLE III
METRICS

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 to [de]compress	s	Measured
T.UUP [T.UDW]	Time to upload [download] the uncompressed file	s	Measured
ET.C [ET.D]	Total energy for [de]compression	J	Measured
ET.UUP [UDW]	Total energy for upload [download] of the uncompressed file	J	Measured
ET.C(0) [ET.D(0)]	Overhead energy for [de]compression	J	ET.C – $I_{idle}\times V_{BS}\times$T.C [ET.D – $I_{idle}\times V_{BS}\times$T.D]
Th.C [Th.D]	[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]
EE.C [EE.D]	[De]compression energy efficiency	MB/J	US/ET.C [US/ET.D]
EE.C(0) [EE.D(0)]	[De]compression overhead energy efficiency	MB/J	US/ET.C(0) [US/ET.D(0)]

The derived throughput captures the efficiency of data transfers from the user's point of view – users produce and consume raw 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.

Energy efficiency. For each compression task with a selected compression level, we calculate the total energy for compression (ET.C) using the method described in Eq. (1) as well as the total energy for decompression (ET.D). For each combination of a compression utility and a compression level, three measurements are conducted and the average energy is calculated. Instead of reporting the energy directly in joules, we report the energy efficiency (EE.C and EE.D) in megabytes per joule (EE.C=US/ET.C and 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).

B. Experiments

We consider two typical usage scenarios as illustrated in Fig. 4. The first experiment (LOCAL) involves measuring the time and energy of compression and decompression tasks performed locally on the smartphone. To eliminate latencies and energy overheads caused by writing files to the internal flash memory, the output of the compression and decompression tasks is re-directed to the null device (/dev/null) that discards all data written to it.

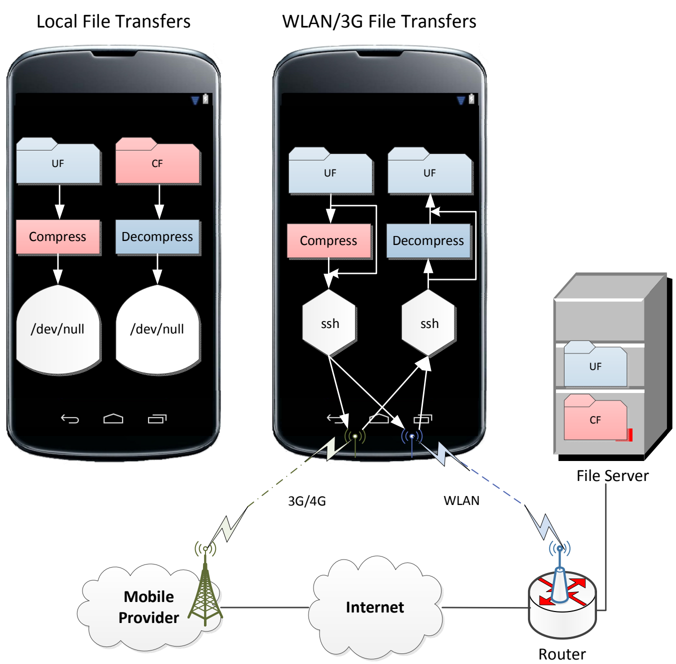


Fig. 4. Data flow of the experiments (blue file icons refer to uncompressed files, red file icons refer to compressed files)

The second group of experiments (WLAN and CELL) involves measuring the time and energy of compression and decompression tasks performed on the smartphone while transferring data to/from a remote server. For the compression tasks, the raw uncompressed input file (UF) is read from the local file system, compressed on the smartphone, and streamed to the remote server over a secure channel. The output files are redirected to the null device of the remote server. For the decompression tasks, the compressed files (CF) are retrieved from the temporary file system of the remote server through a secure channel and decompressed on the smartphone. The output files are redirected to the null device of the smartphone.

The communication between input, compression/decompression, and output operations is carried out through pipes. The execution times include file transfer latencies as well as compression/decompression times. Similarly, the energies are measured for completing the entire chain of tasks. WLAN and CELL correspond to typical file-transfer tasks on smartphones: compressing and uploading files to a remote server, and downloading files from a remote server and decompressing them. In addition to the transfers that involve compression and decompression operations, we evaluate the time and energy needed to upload and download the uncompressed input file over a secure communication channel. In the WLAN experiments, the smartphone connects to the file server through the local router. In the CELL experiments, the file server is reached over the cellular network.

IV. Results

This section discusses the results of our experimental analysis, including the compression ratio (Subsection IV.A), compression and decompression throughput (Subsection IV.B), and energy efficiency as well as the overhead energy efficiency (Subsection IV.C).

A. Compression ratio

Fig. 5 shows the compression ratio on the input dataset of all considered compression utilities and levels. *pigz*'s and *pbzip2*'s compression ratios are equivalent to those of *gzip* and *bzip2*, respectively. Generally, the compression ratio increases with higher compression levels. The best overall compression ratio is achieved by *xz* (4.31 with -9). The lowest compression ratio is achieved by *lzop* (2.07 with -1 through -6 to 2.62 with -9).

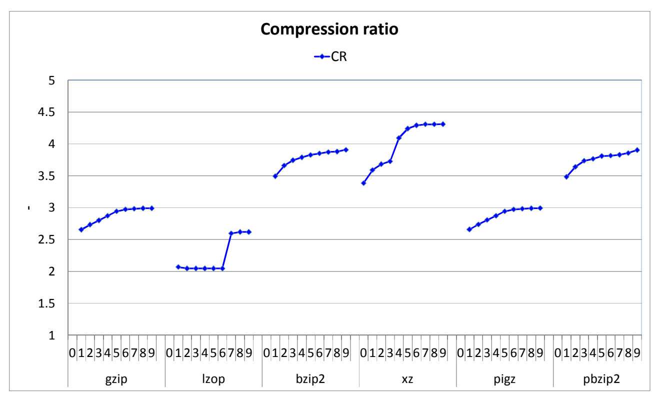


Fig. 5. Overall compression ratio (CR)

B. Compression and decompression throughput

LOCAL. Fig. 6a shows the compression and decompression throughput for *LOCAL*. The compression throughput varies widely across compression utilities and even across compression levels within a single compression utility. The higher compression levels result in lower throughput because of the increased computational complexity. The throughput drop approaches an order of magnitude for *gzip*, *xz*, and *pigz*. The highest compression throughput of ~36.5 MB/s is achieved by *pigz* -1, followed by *lzop* with -1 to -6 with ~30 MB/s. *pigz* fully utilizes four processor cores to almost quadruple the throughput relative to *gzip* (~10.5 MB/s with -1). *xz* and *bzip2* achieve significantly lower compression throughputs (2.3 – 1.6 MB/s for *bzip2* and 2.8 – 0.4 MB/s for *xz*). We observe an almost linear speedup in the *pbzip2* compression throughput relative to *bzip2*.

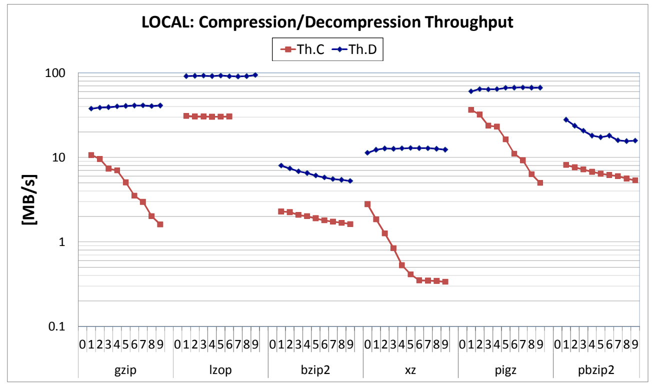
The decompression throughputs are much higher than the compression throughputs. They remain relatively flat or increase slightly with an increase in compression level for *gzip*, *lzop*, *xz*, and *pigz* as the decompression computational complexity remains unchanged but the input files get smaller. Notable exceptions are *bzip2* and *pbzip2*, where the decompression throughputs slightly decrease for higher levels due to increased computational complexity. The highest decompression throughput is achieved by *lzop* (~91 – 94 MB/s), followed by *pigz* (~67 MB/s), *gzip* (~38 MB/s), and *pbzip2* (~28 – 15 MB/s). *xz* and *bzip2* achieve significantly lower decompression throughputs.

WLAN. Fig. 6b shows the measured compression and decompression throughputs for the WLAN experiment. The dashed lines represent the measured effective WLAN throughput when the uncompressed input files are uploaded to the remote server (Th.UUP = 2.33 MB/s) and downloaded from the remote server (Th.UDW = 4.46 MB/s).

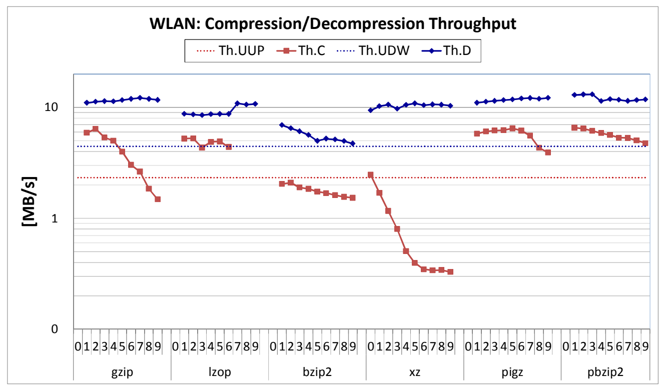
Knowing the network throughputs for uploads and downloads as well as the throughputs in the local experiment and the compression ratios, we can estimate lower and upper bounds for the compression and decompression throughputs of utility *i* and compression level *j*, $Th.C_{(i,j)}^{WLAN}$ and $Th.D_{(i,j)}^{WLAN}$, as shown in Eqs. (3) and (4), respectively. The lower estimates assume that (de)compression on the smartphone and data transfer over the WLAN are not overlapped in time, whereas the upper estimate assumes they are fully overlapped and thus bounded by the network throughput and the compression level. The utilities with a low computational complexity achieve a compression throughput close to the upper bound whereas those with a high computational complexity achieve a compression throughput close to the lower bound. For example, the lower and upper bounds for the compression throughput for *gzip* with -1 are 3.9 MB/s and 6.2 MB/s, and the measured compression throughput is 5.9 MB/s; in contrast, the measured compression throughput for *bzip2* with -1 is 2.04 MB/s and its estimated bounds are 1.8 MB/s and 8.1 MB/s.

The results from Fig. 6b show that compressed data uploads outperform uncompressed uploads for the following combinations: *gzip* with -1 to -7, *lzop* with -1 to -6, *pigz* with -1 to -9, and *pbzip2* with -1 to -9. *bzip2* and *xz* effectively decrease the throughput relative to uncompressed uploads. This result is expected as the compression throughput of *bzip2* and *xz* in the local experiment, $Th.C_{(i,j)}^{LOCAL}$, falls below the WLAN upload throughput, $Th.UUP^{WLAN}$. The best performing utilities are *pbzip2* with -1 at 6.6 MB/s and *pigz* with -5 at 6.5 MB/s, which is over 2.8 times higher than the uncompressed upload throughput.

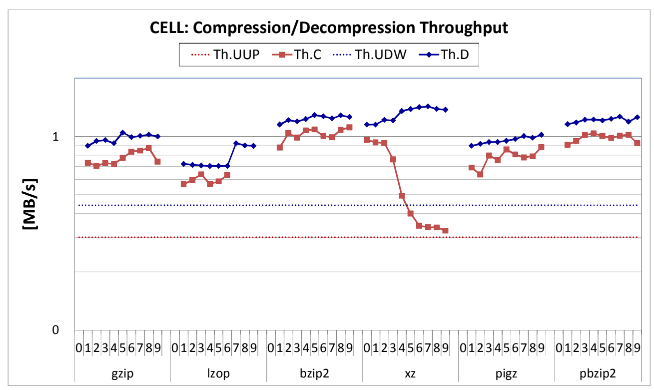
The compressed data downloads improve the effective throughput for all utilities and compression levels relative to the uncompressed downloads ($Th.D^{WLAN} > Th.UDW^{WLAN}$ = 4.46 MB/s). *pbzip2* with -1, *gzip* with -7 to -9, and *pigz* with -5 to -9 all reach ~12 MB/s, which is an almost 2.6-fold improvement in effective download throughput. The decompression throughput of the individual utilities remains fairly constant when changing the compression level. We observe that *pbzip2*, utilizing four processor cores, almost doubles the effective throughput relative to *bzip2*.



A.



B.



C.

Fig. 6. Compression and decompression throughput

$$\frac{1}{\dfrac{1}{CR_{(i,j)}\cdot Th.UUP^{WLAN}}+\dfrac{1}{Th.C_{(i,j)}^{LOCAL}}}\le Th.C_{(i,j)}^{WLAN}\le CR_{(i,j)}\cdot Th.UUP^{WLAN} \quad (3)$$

$$\frac{1}{\dfrac{1}{CR_{(i,j)}\cdot Th.UDW^{WLAN}}+\dfrac{1}{Th.D_{(i,j)}^{LOCAL}}}\le Th.D_{(i,j)}^{WLAN}\le CR_{(i,j)}\cdot Th.UDW^{WLAN} \quad (4)$$

CELL. Fig. 6c shows the compression and decompression throughputs for data transfers over the cellular network interface (3G). The dashed lines represent the measured effective upload and download throughput when transferring the uncompressed input file over the cellular network, $Th.UUP^{CELL}$ = 0.30 MB/s and $Th.UDW^{CELL}$ = 0.44 MB/s, respectively.

The results show that the compressed uploads increase the effective network throughput for all utilities and compression levels. The low network throughput favors utilities with higher compression ratios. Thus, unlike in the WLAN experiments, higher compression levels may help improve the effective throughput. The highest compression throughput of 1.11 MB/s is achieved by *bzip2* with -9, though it performs well with all compression levels (from 0.88 – 1.11 MB/s). Other notable combinations are *xz* with -0 (0.96 MB/s) and *gzip* with -6 (0.83 MB/s). The parallel utilities do not improve the overall throughput because their higher throughputs relative to the sequential utilities far exceeds the network throughput, i.e., $Th.C^{LOCAL} >> Th.UUP^{CELL}$. The lower and upper upload throughput boundaries shown in Eq. (3) hold for cellular data transfers as well. Thus, the effective compression throughput is limited by the network upload throughput and is always below $CR \times Th.UUP^{CELL}$. Note that, although we are faced with limited controllability of the experiments that utilize the cellular network interface due to network provider coverage, utilization of the network at the time the experiments are conducted, and unpredictable Internet latencies, our results are within the expectations defined by Eqs. (3) and (4). Note further that we repeated all measurements multiple times for each utility and compression level and report the averages.

With the low effective throughput for downloads offered by the cellular network ($Th.UDW^{CELL}$ = 0.44 MB/s), all decompression utilities increase the available bandwidth ($Th.D^{CELL} > Th.UDW^{CELL}$) for all tested compression utilities with all compression levels. The bounds for the effective download throughput defined by Eq. (4) apply to downloads over the cellular interface, too. The best performing utility is *xz* with a throughput ranging from 1.15 MB/s (with -1) to 1.43 MB/s (with -7) and *bzip2/pbzip2* ranging from 1.15 MB/s to 1.26 MB/s.

C. Energy efficiency

LOCAL. Fig. 7a shows the energy efficiency of the compression (EE.C) and decompression (EE.D) tasks on the Nexus 4 as well as the energy efficiency when only the energy overhead is considered, EE.C(0) and EE.D(0) (the idle current is assumed to be zero in this case, $I_{IDLE} = 0$).

Expectedly, the energy efficiency for compression varies widely for different utilities and for different compression levels within each utility, especially for *gzip*, *pigz*, and *xz*. The most energy efficient compression utility by far is *lzop* with compression levels -1 to -6. It achieves ~23.5 MB/J when total energy is considered and 35 MB/J when only the energy overhead is considered. A distant second is *pigz* with -1 achieving ~11.5 MB/J total energy efficiency and ~13.5 MB/J overhead efficiency. Following the trends in compression throughputs, higher compression levels for *gzip*, *pigz*, and *lzop* result in a dramatic decrease in energy efficiency. *pigz* and *pbzip2* are more energy efficient than their sequential counterparts when the total energy is considered because they reduce the compression time and thus the relative contribution of the idle energy. *pbzip2* and *bzip2* exhibit low energy efficiencies as does *xz*, the least attractive choice with high compression levels.

The energy efficiency for decompression (Fig. 7a) also varies widely for different utilities. The decompression energy efficiency is relatively stable for individual utilities – it increases slightly for higher compression levels for all utilities except *bzip2* and *pbzip2*. EE.D is ~69 MB/J for *lzop*, ~46 for pigz, ~30 for gzip, and ~8 MB/J for xz. *lzop* emerges as the most energy-efficient choice in spite of its low compression ratio.

WLAN. Fig. 7b shows the energy efficiency for compression and decompression tasks of data transfers over the WLAN interface. In addition, the graph shows the energy efficiency for uncompressed upload (EE.UUP and EE.UUP(0)) and uncompressed download (EE.UDW and EE.UDW(0)). This way, one can easily identify cases when transfers with compression and decompression offer higher energy efficiency than raw uploads (EE.C > EE.UUP) and downloads (EE.D > EE.UDW).

The energy efficiency of uncompressed data upload is EE.UUP = 1.4 MB/J and EE.UUP(0) = 2 MB/J. The energy-efficiency trends are similar to those observed for the effective throughput. The energy efficiency decreases for higher compression levels, especially for *gzip*, *xz*, and *pigz*. Only a small subset of utilities and compression levels improves the energy efficiency relative to the uncompressed upload, including *gzip* with -1 to -6, *lzop* with -1 to -6, and *pigz* with -1 to -8. *xz*, *bzip2*, and *pbzip2* do not offer improvements over the uncompressed transfers at any of the supported compression levels. The most energy efficient approach is to use *gzip*, *pigz*, or *lzop* with the lowest compression level (-1 or -2). For example, *gzip* with -2 achieves an energy efficiency of ~2.9 MB/J, a more than two-fold improvement over the uncompressed upload.

The energy efficiency of the uncompressed data download is EE.UDW = 3.1 MB/J. Decompression using *gzip*, *lzop*, or *pigz* exceeds the energy efficiency of the uncompressed download for both the total energy and the energy overhead. *bzip2* and *pbzip2* are less energy efficient and barely outperform the uncompressed download with low compression levels. The highest energy efficiency of ~8.3 MB/J is achieved when using *pigz* with -6 to -9, which is 2.7 times better than the uncompressed download.

Similar to the estimation of the effective throughput in networked transfers of compressed data, we can also estimate the energy efficiency. By knowing the energy efficiency for uploads and downloads, the energy efficiencies in the local experiment, and the compression ratios, a lower and an upper bound for the compression and decompression energy efficiencies for utility *i* and compression level *j*, i.e., $EE.C_{(i,j)}^{WLAN}$ and $EE.D_{(i,j)}^{WLAN}$, can be computed as shown in Eqs. (5) and (6), respectively. The lower estimates assume that compression/decompression on the smartphone and the data transfer over WLAN are not overlapped in time and the total energy needed is the energy to perform the compression and decompression plus the energy to perform the transfer of the compressed data. The upper estimate assumes full overlap in time and is thus bounded by the energy efficiency of the network transfer and the compression level. The total energy in this case is the sum of the energy overhead (ET.C(0) and ET.D(0)) and the energy for the transfer of the compressed data. Let us consider *gzip -1* as an example: the estimates for the lower and upper bounds for the compression energy efficiency are 2.5 MB/J and 2.8 MB/J, and the measured energy efficiency is 2.8 MB/J. The measured energy efficiency for *bzip2* with -9 is 0.9 MB/J and its estimated bounds are between 0.9 MB/J and 1.2 MB/J. Thus, utilities with a low computational complexity achieve an efficiency close to the upper bound and those with a high computational complexity achieve an efficiency close to the lower bound.

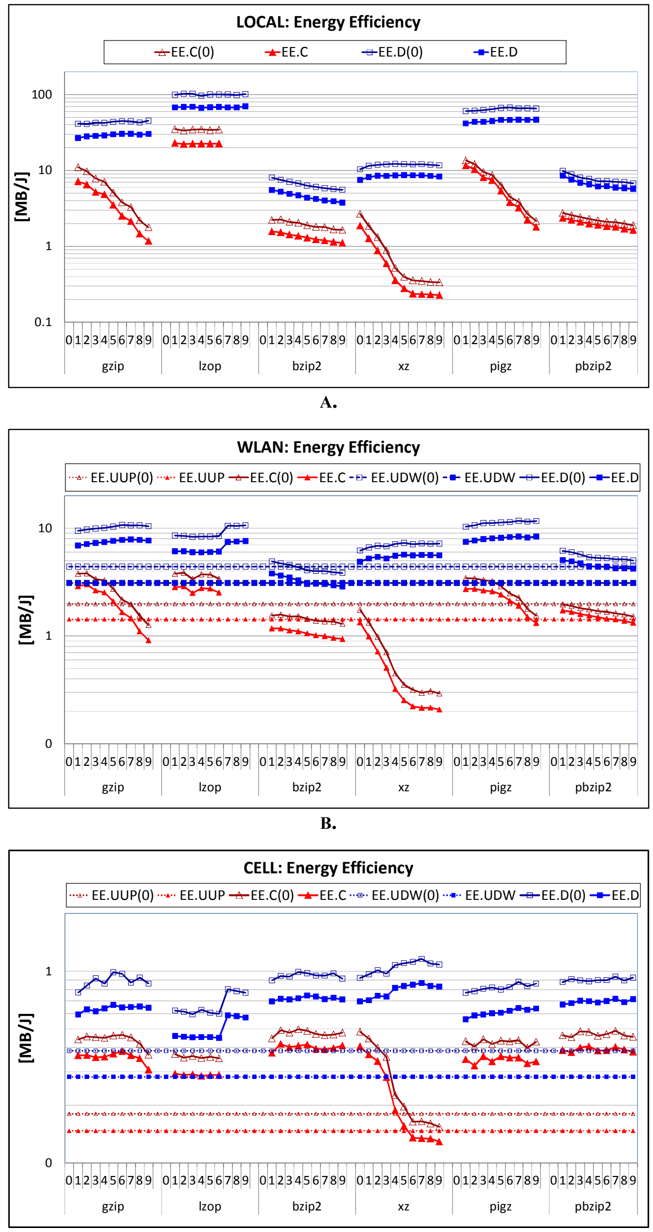
CELL. Fig. 7c shows the energy efficiency of compression and decompression tasks, EE.C and EE.D, respectively, when transferring data over the cellular network. We also show the energy efficiency that considers only the energy overhead, EE.C(0) and EE.D(0). In addition, the graph shows the energy efficiency for uncompressed upload (EE.UUP and EE.UUP(0)) and uncompressed download (EE.UDW and EE.UDW(0)).

The energy efficiency of the uncompressed upload and download are as follows: EE.UUP = 0.15 MB/J and EE.UDW = 0.28 MB/J. Due to this low energy efficiency of raw data transfer, employing any combination of compression utility and compression level offers improvements in energy efficiency, except for *xz* with -6 to -9. The higher computational complexity when increasing the compression level is compensated by the resulting higher compression ratio, which is beneficial in these conditions. Thus, the energy efficiency of *bzip2* and *pbzip2* stays almost constant. In fact, *bzip2* or *pbzip2* offers the highest energy efficiency when uploading data and reaches 0.42 MB/J, a 2.8-fold improvement over the uncompressed transfer.

All decompression options offer an energy efficiency that exceeds the energy efficiency of uncompressed download from the remote server, EE.UDW, which is 0.28 MB/J. Generally, downloading files that were compressed with higher compression levels increases the energy efficiency. The energy efficiency is ~0.65 for *gzip* and 0.7 MB/J for *bzip2/pbzip2*. Interestingly, the best energy efficiency of 0.84 MB/J is achieved by *xz* with -9. *xz* benefits from providing a superior compression ratio in conditions where communication energy dominates the overall energy cost.

$$\frac{1}{\frac{1}{CR_{(i,j)} \cdot EE.UUP^{WLAN}} + \frac{1}{EE.C_{(i,j)}^{LOCAL}}} \le EE.C_{(i,j)}^{WLAN} \le \frac{1}{\frac{1}{CR_{(i,j)} \cdot EE.UUP^{WLAN}} + \frac{1}{EE.C(0)_{(i,j)}^{LOCAL}}} \tag{5}$$

$$\frac{1}{\frac{1}{CR_{(i,j)} \cdot EE.UDW^{WLAN}} + \frac{1}{EE.D_{(i,j)}^{LOCAL}}} \le EE.D_{(i,j)}^{WLAN} \le \frac{1}{\frac{1}{CR_{(i,j)} \cdot EE.UDW^{WLAN}} + \frac{1}{EE.D(0)_{(i,j)}^{LOCAL}}} \tag{6}$$



A.

B.

C.

Fig. 7. Energy efficiency.

V. CONCLUSIONS

Based on the results of our analysis, we provide practical guidelines for selecting the most energy-efficient utilities depending on the usage scenario. For compression tasks, utilities that are fast and have low computational complexity, such as *lzop* with compression levels -1 to -6 and *pigz* with -1, are the most energy-efficient options in spite of their relatively low compression ratios. For decompression tasks, *lzop*, *gzip*, and *pigz* are good choices, as is *xz* when transferring data over a low-throughput wireless network. *The results of our study demonstrate that the common utilities with their default compression levels may not always be the most energy-efficient combinations.* For example, the energy efficiency of compressed uploads over the wireless network using the widely used *gzip* utility with the default compression level -6 is 1.66 MB/J, whereas *gzip* -2 achieves 3 MB/J, an 81% improvement in energy efficiency.

Our findings may guide energy optimizations of data transfers in mobile applications and encourage the development of data transfer frameworks that are conscientious of the mobile device's energy status. For example, a server could easily store multiple copies of the same file, compressed with different utilities and compression levels, to allow the mobile device to choose, based on its capabilities, currently available network bandwidth, energy status, and user preferences, which version of a file to download. Based on similar criteria, the mobile device could choose which format to use for uploading a file, and the server could then convert the file, if necessary, to the best download format(s).

REFERENCES

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

[2] Gartner, Inc., "Market Share: Ultramobiles by Region, OS and Form Factor, 4Q13 and 2013." [Online]. Available: http://www.gartner.com/newsroom/id/2674215. [Accessed: 16-Jun-2014].

[3] Gartner, Inc., "Forecast: PCs, Ultramobiles, and Mobile Phones, Worldwide, 2011-2018, 1Q14 Update." [Online]. Available: http://www.gartner.com/newsroom/id/2692318. [Accessed: 16-Jun-2014].

[4] Cisco, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013–2018." 2014.

[5] "Data Compression Proxy - Google Chrome." [Online]. Available: https://developer.chrome.com/multidevice/data-compression#reduce-data-usage. [Accessed: 30-Oct-2014].

[6] "Onavo," *Onavo*. [Online]. Available: http://www.onavo.com. [Accessed: 30-Oct-2014].

[7] "Mobile Data Compression | App Usage Reporting | Mobile Security | Snappli." [Online]. Available: http://snappli.com/. [Accessed: 31-Oct-2014].

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

[9] S. Kwong and Y. F. Ho, "A statistical Lempel-Ziv compression algorithm for personal digital assistant (PDA)," *IEEE Transactions on Consumer Electronics*, vol. 47, no. 1, pp. 154–162, Feb. 2001.

[10] 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*, 2013, pp. 126–127.

[11] A. Milenkovic, A. Dzhagaryan, and M. Burtscher, "Performance and Energy Consumption of Lossless Compression/Decompression Utilities on Mobile Computing Platforms," in *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on*, 2013, pp. 254–263.

[12] 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.

[13] 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.

[14] "Nexus 4 - Google." [Online]. Available: http://www.google.com/intl/all/nexus/4/. [Accessed: 15-Jun-2014].

[15] "CyanogenMod | Android Community Operating System." [Online]. Available: http://www.cyanogenmod.org/. [Accessed: 14-Jun-2014].

[16] "Android Debug Bridge | Android Developers." [Online]. Available: http://developer.android.com/tools/help/adb.html. [Accessed: 14-Jun-2014].

[17] "The gzip home page." [Online]. Available: http://www.gzip.org/. [Accessed: 25-May-2012].

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

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

[20] "bzip2□: Home." [Online]. Available: http://www.bzip.org/. [Accessed: 25-May-2012].

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

[22] "XZ Utils." [Online]. Available: http://tukaani.org/xz/. [Accessed: 25-May-2012].

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

[24] "pigz - Parallel gzip." [Online]. Available: http://zlib.net/pigz/. [Accessed: 25-May-2012].

[25] J. Gilchrist, "Parallel BZIP2 (PBZIP2)." [Online]. Available: http://compression.ca/pbzip2/. [Accessed: 25-May-2012].