# SMC: Smart Media Compression for Edge Storage Offloading

Ali E. Elgazar, Mohammad Aazam, and Khaled A. Harras
*Carnegie Mellon University*

## Abstract

With the pervasiveness and growth in media technology, user-generated content has become intertwined with our day-to-day life. Such advancements however, have enabled the exponential growth in media file sizes, which leads to shortage of storage on small-scale edge devices. Online clouds are generally a potential solution, however, they raise privacy concerns, are not fully automated, and do not adapt to different networking environments (rural/urban/metropolitan). Distributed storage systems rely on their distributed nature to combat concerns over privacy and are adaptable to different networking environments. Nevertheless, such systems lack optimization via compression due to energy concerns on edge devices. In this work, we propose Smart Media Compression (SMC) for distributed edge storage systems. SMC dynamically adjusts compression parameters, in order to reduce the amount of needless compression, thus reducing energy consumption while providing smaller user file access delays. Our results show an improvement in average file access delay by up to 90%, while only costing an additional 14% in energy consumption.

## 1 Introduction

With ongoing advances in media technologies, user generated content is exponentially growing, giving rise to media storage challenges on user mobile and edge devices [1]. These challenges have prompted most users to resort to utilizing online clouds. However, major centralized storage clouds (CSCs) such as Apple's iCloud, Dropbox, and Google Drive, have come under fire due to multiple hacks, compromising user privacy [2–5]. Given the rise of such privacy issues in CSCs, distributed edge clouds (DECs) became an alternative solution to CSCs. DECs such as Sia, Symform, Beekup, and EdgeStore rely on their distributed nature in order to alleviate privacy concerns [6–9]. While the aforementioned systems provide the user with increased privacy and additional storage, they fail to reduce the average time taken for a user to retrieve an offloaded file and access it. This is mainly due to energy concerns such as that incurred due to file compression [10, 11].

In this work, we introduce an energy efficient compression mechanism, Smart Media Compression (SMC), which can be integrated with any DEC (Section 2). SMC adopts an architecture compatible with DECs and centralized clouds. We show that such compression mechanisms are more beneficial in environments with low bandwidth, which centralized clouds cannot effectively support due to their typical reliance on stable networking connections. SMC is composed of two components, File Popularity Classifier (FPC), and File Compression Mechanism (FCM). FPC automatically classifies files as popular, semi-popular, and unpopular, based on user access patterns in order to identify which files should be compressed and offloaded. Users typically prefer their popular files intact, and unpopular files offloaded. However, FPC identifies a third set of files as semi-popular which are not frequently accessed, but based on temporal locality, may be access again in the close future. Those files are compressed and kept on a user's device.

Using FPC classifications, FCM intelligently decides which media files to compress, and how to compress them. FCM selectively compresses files based on the user's storage requirements and usage, reducing the amount of compression needed to achieve the same results, and as such, reduces overall energy consumption. FCM uses simple lossy FFmpeg re-encoding to reduce the quality of media files. This allows both popular and semi-popular files to exist on the user's device, while still meeting the user's storage requirements. SMC operates like traditional mobile storage offloading and caching. Due to its easy-to-integrate architecture, it enables mobile offloading platforms that do not support caching to have lower average delays on retrieving offloaded files, by increasing local hits on a user's device. Furthermore, typical caching solutions store files as-is in limited quantity, while SMC allows extremely large quantities of files to be kept through compression.

We evaluate integrating SMC with DECs by utilizing a recently developed sample DEC, EdgeStore (Section 3). We attach SMC to EdgeStore (SMC) and compare its performance to the default EdgeStore (ES). Our results show that compression is much more important when the user demands a large portion of their storage offloaded, and its impact is noticed the most in environments with poor network infrastructure. SMC exhibits an improvement in file access delays by 28%, 61%, and 90% in metropolitan, urban, and rural environments respectively. While this improvement comes at the cost of an additional 14% battery consumption, this consumption would be upwards of 43% without the smart checks explained in Section 2.

## 2 Smart Media Compression (SMC)

In this section, we discuss the architecture and functions of SMC, and how it integrates with DECs. Note that the DEC to which SMC is attached, must implement any network adaptability, since SMC does not do so itself. SMC merely provides larger benefits in terms of access delays to solutions which operate in challenged networking environments.

### 2.1 SMC Architecture

Fig. 1 shows the architecture of SMC and how it is integrated with a DEC. Typically, DECs have an interface that takes user input regarding which files to offload and how much of the user's storage the user wants to offload. In a DEC augmented by SMC, SMC components are inserted between the user input and the DEC interface. This enables SMC to dictate to the DEC which files should be offloaded and which should be compressed and kept based on the user's requests and media file access patterns.

SMC consists of two major components: 1) File Popularity Classifier (FPC) which utilizes user access patterns to classify files as popular/unpopular/semi-popular. 2) File Compression Mechanism (FCM) which utilizes the FPC classifications to determine which and how files are to be compressed, and which files are to be offloaded based on the amount of storage the user wishes to free from their device.

### 2.2 File Popularity Classifier (FPC)

We utilize FPC in order to identify which files should be offloaded, which files should be compressed, and which files should remain intact. Our file popularity classifier builds upon Pattern Based Popularity Assessment (PBPA) [12]. PBPA is an algorithm used to classify files in a binary fashion as popular/unpopular. By using temporal locality, we augment PBPA to classify files in a tertiary fashion by introducing semi-popular category as well.

#### 2.2.1 PBPA Overview

PBPA is an algorithm developed to classify file popularity, in order to determine which files should be kept or offloaded from a user's device, and it classifies files over three steps [12].

**Identifying base unpopular files:** In this step, all files that have not been accessed recently (a dynamic parameter based on the user's mobile activity) are considered unpopular, and are the most favored option for offloading.

**Identifying base popular files:** In this step, in the set of active files, the top most accessed portion is considered popular and is the most favored for keeping on the user's device.

**Classifying the remaining unknown files:** PBPA establishes a few metrics which are utilized to gauge a user's unique access pattern to a file [12]. Using these metrics, the access patterns of the unknown files are matched to the access patterns of base popular files. Unknown files that match the base popular files are considered popular, files that do not are considered unpopular.

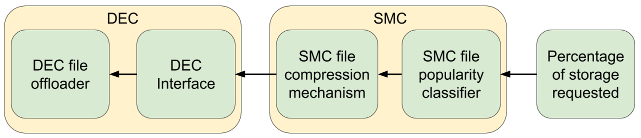

DEC
DEC file offloader
DEC Interface
SMC
SMC file compression mechanism
SMC file popularity classifier
Percentage of storage requested

Figure 1: SMC's architecture attached to a DEC

#### 2.2.2 Augmented PBPA

We create an algorithm, Augmented PBPA (APBPA), which takes advantage of file temporal locality. APBPA Classifies files into three categories instead of two by modifying the final step of PBPA (classifying the remaining unknown files). In the final step of PBPA, files that do not match the access patterns of the base popular files are considered semi-popular instead of unpopular. Although semi-popular files are neither accessed frequently nor display popular access patterns, they were recently accessed and based on temporal locality they are likely to be accessed again [13]. As such, these files can be compressed and kept on a user's device.

#### 2.2.3 Benefits of Tertiary Classification

To measure the benefits of tertiary classification versus binary classification, we utilize real life mobile device file access traces acquired from [14]. These access traces range from 2 to 12 weeks long, with the average trace being 8 weeks long. We classify files using both APBPA and PBPA on a snapshot of each trace at its fourth week (roughly half way through each trace). We then calculate how many files classified as unpopular using PBPA are accessed in the remaining trace, versus how many were classified as unpopular using APBPA. Using PBPA we see that 12% of files classified as unpopular are accessed, while using APBPA we see that only 4% are accessed, giving us a 67% improvement in classifying files.

### 2.3 File Compression Mechanism (FCM)

FCM is the main entity in SMC interfacing with DECs and responsible for deciding which files are compressed, how files are compressed, and which files are offloaded.

#### 2.3.1 How to Compress Files?

In FCM, we utilize the popular library FFmpeg in order to compress media files [15]. We compared FFmpeg to other notable compression libraries such as Video Compressor used by popular messaging app Telegram [16]. We found that FFmpeg has an associated energy cost of roughly 800 Joules per 100MB of media compressed, while Video Compressor costs roughly 1730 Joules per 100MB. This cost comes at a difference in quality of the compressed files, and while Video Compressor produced higher quality compressed files as most messaging apps require, it consumed much more energy. Thus, libraries such as Video Compressor used in social media/messaging apps are not suitable for large scale compression of user data.

Note that when referring to file compression, we refer to lossy compression. As such, compressed media files lose qual-

$$CR_1 = \frac{TS - SR - PF}{SPF}$$

(a) SPF compression ratio

$$CR_2 = \alpha \times \frac{TS - SR}{PF + SPF}$$

(b) Combined compression ratio

| Variable | Definition |
|---|---|
| TS | Total storage |
| SR | Storage requested |
| PF | Popular files |
| SPF | Semi-popular files |

(c) Variable definitions

Figure 2: Compression formulas used given different types of files picked to compress.

**Algorithm 1** Picking files to compress

```
1:  function PICKFILES($popFiles$,$semiFiles$,$unpopFiles$)
2:      $keptStorage$=$totalStorage$ - $requestedStorage$
3:      if size($popFiles$+$semiPopFiles$) > $keptStorage$ then
4:          if size($popFiles$) < $keptStorage$ then
5:              return $semiPopFiles$
6:          else
7:              return $popFiles$ + $semiPopFiles$
8:          end if
9:      end if
10:     return $null$
11: end function
```

ity but are still accessible and view-able by the user without the need for decompression. Furthermore, if a file is compressed and kept on the user's device, the file has its uncompressed version offloaded such that if the user needs a high-definition version of the compressed file, they can always retrieve it.

The most important factor in compression is the compression ratio. In FCM the compression ratio is dynamic, based on how much of the user's storage is to be vacated, and how much is occupied by popular and semi-popular files. However, in order to calculate the compression ratio, first we must decide on which files to compress from the popular/semi-popular files, as compressing only the semi-popular files may not be sufficient if the user wishes to offload large quantities of their storage.

#### 2.3.2 Which Files to Compress?

Alg. 1 describes how we decide which files to compress. If the amount of storage space a user requests to be offloaded falls short of the size of unpopular files, then compression becomes counter-productive and consumes device energy. This occurs when the total amount of storage minus the files the user wishes offloaded (storage saved) is greater than the size of popular and semi-popular files, as such, in this case no compression should take place, set of files compressed is null. This storage requirement check allows our system to compress only when compression is needed, and thus, saves battery consumption. Finally, if the amount requested exceeds the amount of unpopular files, but does not exceed the amount of unpopular files combined with semi-popular files, we compress semi-popular files only. Otherwise we need to compress both semi-popular files as well as popular files.

#### 2.3.3 How Much Should We Compress Files?

The compression ratio (CR) selection varies based on the files being compressed. Fig. 2 shows the different compression ratio formulas given different types of files being compressed.

If we are compressing semi-popular files only, we utilize Fig. 2a (CR1) and set the ratio of compression dynamically to match the difference between the size of popular files and storage saved, such that the compressed semi-popular files fit into that difference.

If we must compress both popular and semi-popular files, we utilize Fig. 2b (CR2). Naturally, we would like to minimally scale popular files in comparison to semi-popular files. As such, we use $\alpha$ in our formula as a weight assigned a different value based on the category of file being compressed. If the file being compressed is a semi-popular file, then $\alpha = PF/SPF$, however, if the file is popular, then $\alpha = SPF/PF$.

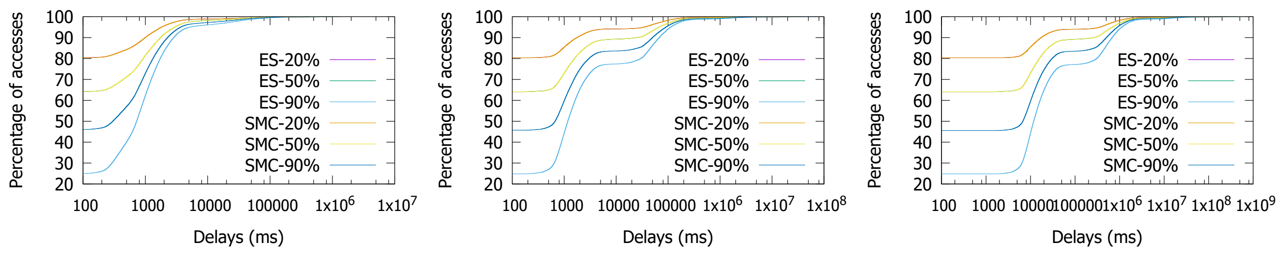
Using the compression ratio, we scale the spacial resolution of media files in order to allow said files to exist within the storage requirements. Note that temporal resolution of video and audio files are not altered. As such, the objective of FCM is to allow files that fall in the popular and semi-popular categories, regardless of their quantity, to exist on the user's device while still meeting the user's storage requirements.

## 3 Evaluation

In this section, we discuss our evaluation setup and present our results. To show the impact of Smart Media Compression on DECs, we attach SMC to a recently developed DEC, EdgeStore, and we compare the results between default EdgeStore (ES), and SMC integrated with EdgeStore (SMC).

### 3.1 EdgeStore Overview

EdgeStore is a private deployable DEC platform [9]. EdgeStore allows users to utilize their household devices such as desktops, mobile devices, tablets, IoTs, so on, as a private distributed storage cloud where they can automatically offload files, taking advantage of the underutilized storage most users have between their household devices. EdgeStore supports automatic offloading of user files based on user access patterns, as such, it is a perfect candidate for evaluating SMC, since SMC also relies on user access patterns for file compression.



(a) Metropolitan environment (b) Urban environment (c) Rural environment

Figure 3: CDF of access delays in every environment using ES and SMC with varying percent of files offloaded.

## 3.2 Experimental Setup

We implement EdgeStore and SMC in java, and utilize the Java-based ONE simulator to simulate device communication in a household scenario [17]. The experiment is based on 90 days of in-simulation time, in which a main mobile device access files. EdgeStore automatically offloads files it deems unpopular during the simulation, while SMC+EdgeStore compresses some files and offloads the others.

In order to account for users experiencing different network conditions, we create three separate simulation environments addressing different networking infrastructures (rural, urban, and metropolitan). In the rural environment, devices communicate using only direct techniques (Bluetooth, ad-hoc), while in the urban environment devices communicate using WiFi only. In the metropolitan environment devices utilize WiFi when it is available and 4G network otherwise.

As EdgeStore is a storage solution that utilizes household edge devices, we utilize household mobility models to simulate user node movement. We utilize the same mobility model utilized by previous EdgeStore evaluation.

## 3.3 Dataset

To simulate user accesses of files, we utilize the same set of real life mobile device file access traces utilized in EdgeStore evaluation [14]. The user traces have been filtered to contain only audio, video, and image files that are larger than a particular threshold in size (2MB for audio and video, and 500KB for images). This filtering is done to ensure that the files left in the traces are the ones accessed by users, and not by the system itself (logo images, device sound clips, etc.). In this evaluation, we use the most active data trace from all the mobile data access traces we acquired. This active data trace contains 1417, 181, 18 images, videos, and audios respectively, with a combined number of 112417 accesses to said files over the 90-day trace. The average sizes of images, videos, and audios, are 2.1, 91.4, and 5.4MBs respectively.

During the 90-day simulation, an offloading device is fitted with an access trace, and the offloading/compression decisions are based on the file access patterns for that user. On average, during the simulation the user's popularity distribution was roughly 62%, 37%, and 4% unpopular, semi-popular, and popular respectively with 6%, 7%, and 2% variation.

## 3.4 Metrics

We utilize two main metrics to measure the result of our evaluations, these metrics are as follows:

**Access Delay:** File access delay is the time taken for a file to be presented to the user when the user requests the file. A file access request can either be a local hit (available on the native device) or a miss (offloaded to another edge device). Compression plays a huge factor in access delays as more files are available on the user's system instead of being offloaded.

**Energy Consumption:** Energy consumption is heavily impacted by the amount of files compressed, amount of communications, as well as the communication technology utilized. In our evaluations, we fix the throughput of our 4G, WiFi, and Bluetooth interfaces to 75 Mbps, 40 Mbps, and 20 Mbps respectively, which are typical rates [18]. Our tests showed that each 1MB of data transferred consumes on average 25, 15, and 10 Joules of energy utilizing 4G, WiFi, and Bluetooth respectively. Computations performed by APBPA consume roughly 550 Joules per hour. Compression using FFmpeg consumes roughly 800 Joules per 100MB of media. This testing was performed on an HTC Android device which has a battery capacity of 2600 mAh (at 5V, this is roughly 46800 Joules of energy per full device charge). These values were used to simulate battery consumption throughout the experiment. We note that when users view a compressed file, they always opt to retrieve the original HD version of the file, causing additional energy usage.

## 3.5 Results

In our simulations, we use a single parameter, percentage of storage offloaded as requested by a user. We vary the percentage between 20%, 50%, and an extreme 90%. All other EdgeStore related parameters are set to default values as explored in [9].

**Impact of smart compression checks:** Fig. 3 shows the CDF of delays in every environment given different percentage of storage offloaded. In all three environments, there is no improvement in access delays between ES and SMC when we offload 20% and 50% of files, giving us overlapping lines between ES-20%/SMC-20%, and ES-50%/SMC-50%. This is due to the preventative compression checks implemented in FCM. In the trace, 62% of the files were unpopular on average during the experiment. As such, when the user chooses to

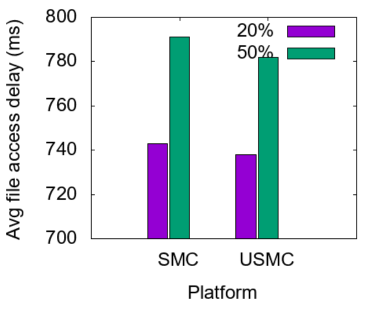


Figure 4: Access delay in SMC versus Unchecked SMC (USMC)

offload less than 62% of files, compression does not benefit the system, as compressing files merely adds more unpopular files to the system which are unlikely to be accessed. In this case, SMC's compression checks prevent wasteful compression and no files are compressed, giving the same result as ES.

**Impact of compression on access delays:** When the user offloads an extreme 90% of their storage, the delays increase significantly as more files are offloaded and thus, need to be retrieved later. However, using SMC, the percentage of instantaneous accesses falls around 47%, compared to 25% when using ES. This increase in instantaneous access causes the average delay to plummet. In the metropolitan, urban, and rural environments, the average delay drops from 312ms to 243ms, 432ms to 267 ms, and 2.1s to 1.1s respectively, giving us a 28%, 61%, and 90% improvement in delays on average, respectively. These results show that while SMC provides an improvement to storage solutions in any environment, SMC is most beneficial in environments with low bandwidth and connectivity, where it is more difficult to retrieve offloaded files, and hence more important to keep them on a user's device.

The improvements from ES to SMC are due to the fact that with tertiary classification and compression, the user's device offloads less files. Those improvements are amplified in urban and rural environments due to connection downtimes. This causes devices using ES to have to wait prolonged periods of times until they can retrieve files, in comparison to SMC where those files were classified as semi-popular and compressed instead. This shows that in the absence of a stable network connection, compressing files and keeping them on a user's device becomes imperative as the downsides of not doing so requires the user to wait for a stable network connection to retrieve a file, and causes the average delays on accessing a file to skyrocket.

**Impact of unchecked compression on average delays:** To showcase the importance of smart compression, we disable the aforementioned checks in a few of our simulations and compare the results using the rural environment. Fig. 4 compares SMC and Unchecked SMC (USMC) with compression checks turned off. We observe little benefit in USMC compared to SMC, with USMC producing an average delay only 0.6 and 1.1% improved when we offload 20 and 50% of files. Not only is the improvement minor without compression checks, but we consume more energy as the device is blindly compressing files without need.

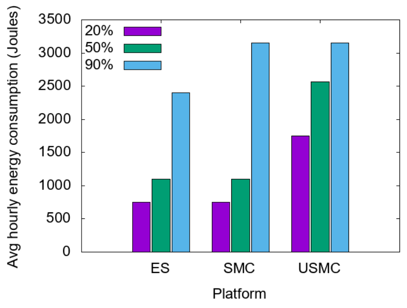


Figure 5: Comparison between ES, SMC, and USMC in energy consumption

**Impact of unchecked compression on battery usage:** Fig. 5 shows the comparison between ES, SMC, and USMC in terms of energy consumption during the simulation. Energy drain is bound to occur in our system whether ES or SMC is utilized, due to the large quantities of files being offloaded. During the simulation, there is no difference between ES and SMC when we offload less than 50% of files in terms of energy consumption. In comparison, USMC has a much higher energy consumption for the same amount of files offloaded. As we can see, when we offload an amount of files more than the amount of unpopular files (90%), the energy consumption of SMC is the same as USMC as the checks only apply to the aforementioned specific case. The average hourly energy consumption in all simulations for ES, SMC, and USMC is 1415, 1660, and 2490 Joules respectively. As such, devices running ES, SMC, and USMC ran out of battery every 33, 28, and 19 hours respectively. While SMC costs an additional 14% of energy consumption, it prevents an additional 43% energy consumption through its smart checks.

## 4 Conclusion and Future Work

In this paper, we discussed the need for a private and automated edge storage systems, and how compression can massively benefit such systems. We showed how tertiary file popularity classification can be more advantageous than binary. Furthermore, we showed how compression must account for dynamic factors, such as the amount of storage the user requests offloaded, and the ratio between popular and semi-popular files in the user's system. Lastly, we showed how important it is for compression to be applied selectively. In the future, we intend to examine pre-fetching techniques to reduce the need for compression, thus reducing energy waste. Additionally, we will investigate the possibility of having variable compression ratios based on the popularity of the files, instead of treating files in the same category as equals.

## Acknowledgement

This publication was made possible by NPRP grant # 8-1645-1-289 from the Qatar National Research Fund (a member of Qatar Foundation). The findings achieved herein are solely the responsibility of the authors.

## 5 Discussion Topic Section

**Feedback and controversial points:** In the last few years, most research in edge/fog computing has been targeted towards computational offloading related challenges to better serve low-latency edge applications. We have been, and in this paper continue to, argue for storage-related problems within the context of edge/fog computing. Our experience has been that this topic receives mixed levels of support to begin with. We anticipate similar debates and feedback in this workshop.

Other criticism or feedback we anticipate is related to the criticism of centralized storage solutions for their security vulnerabilities, and the favoring of decentralized edge storage. Again, in our experience, people tend to argue on what would be more secure and private. Are centralized massive storage solutions that are heavily guarded but a target for many hackers more or less secure than obscure edge ensembles of devices that are less patched up?

Another controversial point would be that while we are providing a solution that reduces the potential energy consumption of compression, our solution still requires more energy consumption overall from any system it is attached to.

**Type of discussion:** This paper will likely generate discussion regarding the benefits and downsides of compression of large amounts of media files on edge devices, which are typically limited in computational power and energy capacities. Moreover, another trigger for a discussion could be efficient and productive use of co-located edge devices for data offloading.

**Open issues not addressed:** In our paper, we do not address the potential for offloading automation using SMC. For example, the DEC utilized in our evaluation, EdgeStore, already supports automation through binary file classification. However, other DECs typically require manual offloading by the user. DEC platforms such as Sia or Beekup might benefit from automation through SMC's tertiary classification and it would be worth studying the impact of automation on such platforms. Furthermore, it may be beneficial to augment our solution such that it accounts for the networking condition itself when deciding which files to compress.

**When does our work fail:** Our presented smart compression relies on accurate classification of file popularity based on user access patterns. In order to have accurate classification, the user must display a consistent access pattern to his files. In some of the access traces we acquired, a couple of users would access their files normally and consistently, then go for periods of time (up to a week) with no activity. These file-access gaps may skew the user's access pattern even though they are rare. Additionally, while rare, family members may exchange devices, and since file classification is based on a user's access patterns, results may be inaccurate when switching devices from one user to another. As such, a new owner of a device might need to wait until the system converges on his new access pattern.

Both aforementioned issues can lead to the compression of popular files instead of semi-popular ones, which can delay access to files which the user expects to have intact. Ultimately, the user can still retrieve file-originals that were offloaded.

## References

[1] G. Shao, "Understanding the appeal of user-generated media: a uses and gratification perspective," *Internet Research*, vol. 19, no. 1, pp. 7–25, 2009.

[2] H. Dinh, A. Dworkin, C. O'Neill, S. Savage, J. Leak, M. Aazam, and M. St-Hilaire, "Omnibox: Efficient cloud storage by evaluating dropbox and box."

[3] "Dropbox hacking," https://tinyurl.com/jzptkee.

[4] "Apple hacking gets worse," http://www.zdnet.com/article/icloud-accounts-breach-gets-bigger-here-is-what-we-know/.

[5] "Google cloud hacking," http://searchengineland.com/after-the-googlehack-33508.

[6] D. Vorick and L. Champine, "Sia: Simple decentralized storage," Technical Report, Sia, 2014, Tech. Rep., 2014.

[7] Y.-Y. Teing, A. Dehghantanha, K.-K. R. Choo, T. Dargahi, and M. Conti, "Forensic investigation of cooperative storage cloud service: symform as a case study," *Journal of Forensic Sciences*, 2016.

[8] F. e. a. Rizzo, "Beekup: A distributed and safe p2p storage framework for ioe applications," in *ICIN 2017*. IEEE, 2017, pp. 44–51.

[9] A. Elgazar, M. Aazam, and K. Harras, "Edgestore: Leveraging edge devices for mobile storage offloading," in *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2018, pp. 56–61.

[10] C. M. Sadler and M. Martonosi, "Data compression algorithms for energy-constrained devices in delay tolerant networks," in *Proceedings of the 4th international conference on Embedded networked sensor systems*. ACM, 2006, pp. 265–278.

[11] T. Ma, M. Hempel, D. Peng, and H. Sharif, "A survey of energy-efficient compression and communication techniques for multimedia in resource constrained systems," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 963–972, 2013.

[12] A. Elgazar, K. Harras, M. Aazam, and A. Mtibaa, "Towards intelligent edge storage management: Determining and predicting mobile file popularity," in *2018 MobileCloud*. IEEE, 2018, pp. 23–28.

[13] S. Jiang, X. Ding, F. Chen, E. Tan, and X. Zhang, "Dulo: an effective buffer cache management scheme to exploit both temporal and spatial locality," in *Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies*, vol. 4, 2005, pp. 8–8.

[14] R. Friedman and D. Sainz, "File system usage in android mobile phones," in *9th SSC*. ACM, 2016, p. 16.

[15] Ffmpeg. Retrieved on (2019-3-19). [Online]. Available: https://www.ffmpeg.org/

[16] Telegram video compressor. Retrieved on (2019-3-19). [Online]. Available: https://github.com/lalongooo/VideoCompressor

[17] A. Keränen, J. Ott, and T. Kärkkäinen, "The one simulator for dtn protocol evaluation," in *STT*. ICST, 2009, p. 55.

[18] Throughput comparison. Retrieved on (2019-3-19). [Online]. Available: http://www.bandwidthplace.com/internet-speed-test-3g-4g-lte-and-wifi-who-wins-article/