Skyfiles: Efficient and Secure Cloud-assisted File Management for Mobile Devices

Ying Mao Jiayin Wang Bo Sheng
Department of Computer Science
University of Massachusetts Boston
{yingmao, jane, shengbo}@cs.umb.edu

***Abstract*—This paper targets the application of cloud storage management for mobile devices. Because of the limit of bandwidth and other resources, most existing cloud storage apps for smartphones do not keep local copies of files. This efficient design, however, limits the application capacities. In this paper, our goal is to extend the available file operations for cloud storage service to better serve smartphone users. We develop Skyfiles, an efficient and secure file management system that supports more advance file operations. Our basic idea is to utilize cloud instances to assist file operations. Particularly, Skyfiles supports download, compress, encrypt, convert operations, and file transfer between two smartphone users' cloud storage spaces. In addition, we design protocol for users to share their idle instances.**

I. Introduction

With recent advances, smartphones have become one of the most revolutionary devices nowadays. According to Nielsens report made in March 2012, about 50.4% of U.S. consumers own a smartphone. Consequently, a variety of applications have been developed to meet users' demands in all aspects (both Apple Store and Google Play hit 1 million apps in 2013). Today's smartphones have gone far beyond a mobile telephone as they have seamlessly dissolved in people's daily life.

In this paper, we consider the smartphone application of cloud storage service which is another recent emerging technology. Representative products include iCloud [1], Dropbox [2], Box.com [3], Google Drive [4], and others [5], [6]. Basically, each user holds a certain remote storage space in cloud and can access the files from different devices through the Internet. Synchronization and file consistence are guaranteed in these cloud storage services. When smartphones become popular, it is ineluctably for users to couple cloud storage service with their smartphones. However, users and developers have encountered specific challenges due to the limitations of smartphones. First, the storage capacity of a smartphone is limited compared to regular desktop and laptops. Second, the network bandwidth of the cellular network is limited. At this point, major U.S. mobile networks carriers rarely provide unlimited data plans and the service scalability is limited by fundamental constraints. Finally, energy consumption is a critical issue for smartphone users. With the above constraints in mind, most existing smartphone apps for cloud storage service follow one important design principle of not keeping local copies of the files stored in cloud because smartphones may not have sufficient space to hold all the files, and downloading those files consume a lot of bandwidth and battery power. Instead, only meta data is kept on smartphones by default. Though this design is efficient, it limits the capabilities of the apps. Some file operations that can be easily done with local copies become extremely hard, if not impossible, for smartphone users, e.g., compressing files and transferring files to another user.

In this paper, we develop Skyfiles system for smartphone users to manage their files in cloud storage with more capabilities. Our basic idea is to launch a cloud instance to assist users to accomplish some file operations. It is motivated by the fact that the cloud instance is inexpesive and sometimes free. For example, Amazon Web Service (AWS) [7] provides 750 free Micro instance hours per month. By using the resources of the instance, smartphone users will significantly reduce the bandwidth consumption for file operations. Skyfiles does not require users to keep local copies of files, but possesses the following new features: (1) It extends the available file operations for mobile devices to a more enriched set of operations including download, compress, encrypt, and convert operations. (2) It includes a protocol for two smartphone users to transfer files from one's cloud storage space to the other's cloud storage. (3) It includes secure solutions for all the above operations to use shared cloud instances, i.e., instances created by other users. The rest of this paper is organized as follows. Section II overviews the related work and Section III introduces background information about cloud storage service. In Section IV, we present the basic solution for cloud-assisted file operations and file transfer between users. Section V includes more efficient solutions that allow users to use shared instances. We evaluate the performance of Skyfile in Section VI and conclude in Section VII.

II. Related Work

One category of related prior work focuses on offloading computation for mobile devices. MAUI [8], Cuckoo [9] and ThinkAir [10] implement an Android framework on top of the existing runtime system. They provide a dynamic runtime system which can decide whether a part of an application should be executed on the mobile device or a remote server. Our work in this paper targets on file operations rather than general computation, and our main objective is to save the network bandwidth for mobile devices.

Besides offloading computation, SmartDiet [11] aims at offloading communication-related tasks to cloud in order to

save energy of smartphones. Wang et. al [12] investigate Dropbox users to understand characteristics of personal cloud storage services on mobile device. Their results show possible performance bottlenecks caused by either the current system architecture and the storage protocol. In [13], the authors focus on the impact of virtualization for Dropbox-like cloud storage systems. Another related area in the prior work is to use cloud to enhance mobile device security and privacy. CloudShield [14] presents an efficient anti-malware tool for smartphones with a P2P network on the cloud. Confidentiality as a Service (CaaS) [15] paradigm is proposed to provide usable confidentiality and integrity for regular users with little training on security. CaaS separates capabilities and requires less trust from cloud or CaaS providers, and performs automatic key management.

III. Background

This section introduces background information about cloud storage service. Most of our experiments in this paper are conducted on Dropbox platform. Thus, this section regards Dropbox as a representative service provider and briefly describes the Dropbox architecture and functions for user applications. As a leading solution in personal cloud storage market, Dropbox provides cross-platform service based on Amazon Simple Storage Service(S3) for both desktop and mobile users. The architecture of Dropbox follows a layered structure. At bottom level, Amazon S3 infrastructure provides a basic interface for storing and retrieving data. The above layer is Dropbox core system which interacts with S3 storage service and serves higher level applications. The top layer is the official Dropbox application and a set of APIs for developers to build third party applications.

To manage third party applications, Dropbox assigns each of them an unique *app key* and *app secret*. When a user launches an application, Dropbox server follows the OAuth v1 [16] for authentication. When launched by a user, the third party app contacts the Dropbox server to obtain a one time request token and request secret. Then, the app uses them to form a redirect link and presents the link to the user. When accessing this link, the user will be prompted to login with his Dropbox account and the Dropbox server will verify the redirect link and the user's login information. After a successful login, the server will return an *access token* and *access secret* to the application, which grant access permissions on the user's data.

IV. Basic Solutions

In this section, we first describe the framework of Skyfiles and cloud-assisted file operations. Then, we focus on the file transfer between two users' cloud storage in Skyfiles. Our objective is to accomplish file operations with minimum network bandwidth assumption.

A. Framework and Basic Cloud-assisted Operations

In Skyfiles, each mobile device is associated with a cloud storage account. Similar to other related apps, Skyfiles by default does not keep local copies of the files stored in cloud because of the storage limit and bandwidth consumption. Instead, Skyfiles maintains a *shadow file system* on the mobile device which includes the meta information of the files stored in cloud. This local file system is built on service provider's APIs and synchronized with the cloud storage.

Skyfiles supports two categories of file operations. The first category is the basic file operations that are commonly available in service providers' APIs such as creating/deleting/renaming a file. The second category is a new set of advance file operations that require assistance from a cloud instance. Skyfiles recognizes the category each operation request from a user belongs to and processes it accordingly. The first category will be handled by regular API function calls. For the second category, Skyfiles will create a cloud instance and then forward the file operation request to the instance for processing. In particular, we have designed the following four cloud-assisted operations in Skyfiles:

Download: This operation allows a user to download files directly to his cloud storage. Given the location of the target files such as URLs, the conventional way of downloading is to first obtain the files on user devices and then synchronize with/upload to the cloud storage. In Skyfiles, the cloud instance will fetch the files and then upload them to the user's cloud storage. Thus the downloading and uploading will not consume mobile device's bandwidth.

Compress: This operation enables a user to compress existing files or directories stored in cloud. If user devices hold local copies of the target files, the operation can be easily accomplished and the generated compressed file can be uploaded to the cloud storage. In Skyfiles or other similar apps for mobile devices, however, the actual file contents are not available. Thus we design an interface that the user can select the target files based on the local shadow file system with meta data and then the compressing operation is forwarded to a cloud instance for execution. The instance will fetch the specified files from the cloud storage, compress them, and upload the compressed file back to the cloud storage.

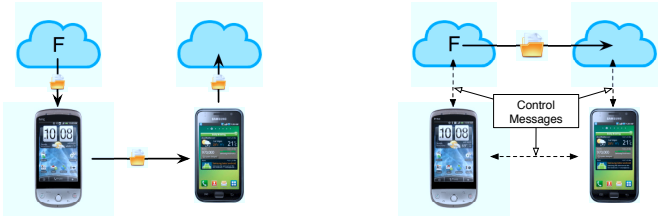
Encrypt: This operation is similar to compress and does not exist in current apps that do not keep local file copies. In Skyfiles, the user can choose the target files and the cipher suite including the cryptographic algorithm and key. The encryption operation will be sent to a cloud instance. Similarly, the cloud instance will download the target files from the user's cloud storage, encrypt them, and send the ciphertext back to the cloud storage.

Convert: The last operation is particularly for media files such as pictures and video clips. When a user wants to view a picture stored in cloud, he has to download it to his smartphone. Nowadays, high-resolution picture files could be very large, but a smartphone user may not benefit from it because of the limited screen size. In Skyfiles, therefore, a user can specify an acceptable resolution when viewing a picture and the request will be processed by a cloud instance. The original picture will be downloaded to the instance and then converted to a smaller file according to the user-specified resolution. Finally, the converted picture is sent to the user.

Overall, we develop the above set of advance file operations for mobile devices which are impossible to achieve without local file copies. In Skyfiles, a cloud instance is launched to assist a user to accomplish these file operations. During the execution of the file operations, the cloud instance will periodically send heartbeat messages to the smartphone to report the progress and status. The smartphones only consumes a small amount of bandwidth for exchanging control messages with the cloud storage server and the cloud instance.

B. File Transfer between Users

In this subsection, we present an important feature of Skyfiles which allows file transfer between users. While most cloud storage services allow a user to share files with another user, copying files across different user spaces is not supported. However, file sharing between users cannot substitute file transfer (make a copy). With file sharing, a user's actions on the shared files will affect other users. For example, if a user deletes the shared files, all the other users lose those files too. In this subsection, we consider the file transfer between user spaces illustrated in Fig.1. Assume two users carrying smartphones with data plan subscription meet with each other and both have storage spaces in cloud. One user (as the sender) wants to transfer files in his cloud storage to the other user's (as the receiver) cloud storage. We aim to develop an efficient solution to support this feature with as little network bandwidth consumed as possible. In the conventional solution, the sender can download the target files to his smartphone and send them to the receiver's phone through Internet or short range connection such as Bluetooth and NFC. Upon receiving the files, the receiver's phone can upload or synchronize them with the cloud storage. This solution, however, is not efficient in terms of bandwidth consumption, especially when the target files are large, e.g., bunch of pictures and video clips, as the sender has to download the entire files and the receiver has to upload all the files.



(a) Conventional Solution (b) Skyfiles

Fig. 1: File Transfer between Two Users

In Skyfiles, we solve the problem by following the same design principal that utilizes a cloud instance to assist users to transfer files between their cloud storage spaces. Basically, a cloud instance is initialized and plays a role of relay node. It fetches the target files from sender's cloud storage space and then forwards them to the receiver's cloud storage. In this means, both sender and receiver's smartphones do not have to hold a local copy of the target files and the bandwidth is consumed only by control messages between smartphones and the cloud instance/cloud storage server. The basic design of using a cloud instance, however, is challenging in practice when no trust has been established between the sender and receiver. The cloud instance can be created by either the sender or receiver. In either case, it is not a secure solution for the party who does not own the instance because the cloud instance will need to obtain the security credentials of cloud storage from both sender and receiver to complete the file transfer. Therefore, the owner of the cloud instance will be able to access the cloud storage space of the other user which could breach data privacy and lead to other malicious operations.

Algorithm 1 File Transfer from U_A to U_B

1: U_A starts a cloud instance I_A
2: $U_A \rightarrow I_A$ (cellular network) : U_A's security credentials to access his cloud storage space, source file location F_{src}, and intermediate file location URI_F
3: I_A downloads F from U_A's cloud storage to and stores it at URI_F
4: $U_A \rightarrow U_B$ (NFC) : URI_F (file location on I_A)
5: U_B starts a cloud instance I_B
6: $U_B \rightarrow I_B$ (cellular network) : U_B's security credentials to access his cloud storage space, URI_F, and destination file location F_{dst}
7: I_B copies F from I_A (URI_F) to its local storage
8: I_B uploads F to U_B's cloud storage space (F_{dst})

To address the above issue, in Skyfiles, we develop a solution that requires a cloud instance from both sender and receiver. In particular, we implement the following Algorithm 1 to accomplish the file transfer. Assume user U_A is trying to send a file F (F can also represent a set of files) to user U_B. Let F_{src} be the location of F in U_A's cloud storage and F_{dst} be the destination location that U_B will put in his cloud storage. In our protocol description, U_A and U_B also represent the users' smartphones. The following Algorithm 1 presents the major steps for file transfer. First, U_A starts a cloud instance (I_A) and uploads the security credentials for accessing his cloud storage space to the instance. U_A's request also includes the source file location F_{src} and an *intermediate file location* URI_F (uniform resource identifier) which indicates where to store F on the instance. The cloud instance will use the security credentials to download the target file F to its local disk. At this point, I_A needs to make F accessible to user U_B. It first sends U_A the intermediate file location URI_F. Then, I_A can set F publicly available or create a guest account and set the permissions of F so that only the guest account can access it. In the latter case, the security information for logging as the guest account, such as login password or identity file, needs to be sent back to U_A as well. After that, the steps on U_A's side have been completed. Then, U_A needs to notify U_B necessary information for accessing F. Since this step of communication includes sensitive information, Skyfiles adopts NFC protocol to securely deliver URI_F and optional login information from U_A's smartphone to U_B's phone. At receiver's side, U_B also starts a cloud instance I_B which obtains F from I_A based on URI_F. Finally, I_B uploads F to F_{dst}. Here, both sender and receiver start a cloud instance to behave as their agents.

The data transfer of F is between cloud instances and cloud storage servers which does not consume bandwidth of users' smartphones. Meanwhile, the security credentials for accessing cloud storage are only sent to the instance created by the same owner. Thus, in Skyfiles, file transfer between two users' cloud storage is efficient and secure.

V. Solutions with Shared Cloud Instances

The solutions presented above efficiently enable smartphone users to manage their files in cloud by launching cloud instances for assistance. In practice, however, the following two issues may hinder the deployment of the proposed solutions. First, initializing a cloud instance incurs a significant overhead, e.g., the overhead ranges from 15 seconds to 30 seconds in our experiments in SectionVI. The second issue is the cost of launching cloud instances. Although cloud service is inexpensive, frequently starting cloud instances may still increase the cost of users.

In this section, we propose an enhancement for Skyfiles that solves the overhead and cost issues by allowing users to share cloud instances with each other. It is motivated by the fact that cloud service providers charge the instance service at a certain time granularity. For example, most providers [7], [17], [18] charge the usage of cloud instance at the granularity of an hour. For regular file operations, it is an excessive time period. When starting a cloud instance for file operations, a user does not have to terminate the instance when the operation is done. The instance can be kept running until an additional cost is about to be charged. For example, assume a service provider charges the instance service in the time unit of an hour, when a user starts an instance and finishes his file operations in the first 5 minutes, the instance can stay active for another 55 minutes without extra cost. During this idle time period, the instance can serve other users or other file operations from the same user. While benefiting the performance, the design of sharing instances among users incurs challenge for security. First, it is risky for a user to upload his security credentials of cloud storage account to other users' cloud instances. The owner of the instance may monitor and catch the security credentials, and gain the access to the user's cloud storage space. Second, when open to public, the shared cloud instances may be used by malicious users to launch attacks.

In Skyfiles, we have developed a framework for sharing instances which requires a trusted server. This server maintains a list of available cloud instances for sharing and coordinates the users who request instances and share instances. Skyfiles applies the following two basic policies to address the security concerns. First, an instance is shared in the form of launching a background service and accepting requests from other users, rather than allowing other users to login and execute arbitrary programs. Second, when using a shared instance, a user does not upload the security credentials of his cloud storage account in plaintext, but in an encrypted format. In this way, the owner of the instance can not gain the access to the tenant user's cloud storage space and any user's privileges on shared instances are limited to the specified file operations.

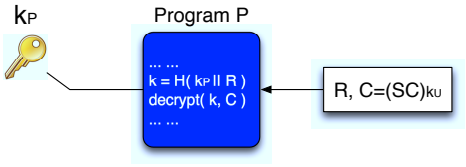


Fig. 2: Structure of Service Program P: secret k_P is embedded in P which can generate k_U and decrypt ciphertext C.

Specifically, there are three types of entities in our design, a trusted server S, a user U_A who wants to conduct file operations on a shared instance, and an available cloud instance I owned by another user U_B. The trusted server holds a binary program P that can be running on a shared instance to provide Skyfile services to other users. Once a user (U_B) decides to share his instance (I), the instance will contact the server and forward the basic information about I such as operating system, hardware setting, and the time left for sharing. The response from the server is an executable binary P, where a secret key k_P is embedded. We assume k_P is protected by program obfuscation techniques and I_B is not able to derive k_P from P (see Fig. 2). In addition, the server will periodically change k_P and re-compile the binary. Upon receiving P, I will execute P as a service and be ready to accept other users' requests. The server, on the other hand, adds I_B into the list of available instances for sharing. Finally, each shared instance I can set a scheduled task to automatically shut down the instance before the additional charge is incurred. During the shutdown process, I also notifies the server S which will consequently remove I from the list of available instances for sharing.

$I \rightarrow S$: willing to share;
$S \rightarrow I$: program P embedded with a key k_P;
I executes P;
S adds I into the list of available instances

Single User Operations: When a user (U_A) requests to use a shared instance to conduct operations on his files in cloud, he needs to first contact the trusted server S. S will generate a random seed R_A for the requesting user and apply a hash function H on k_P and R_A to generate another key k_A, $k_A = H(k_P||R_A)$. The server then chooses a shared instance from the list to serve U_A and it could be an interactive process that involves U_A's opinion. Assume I is selected, the server sends $\{R_A, k_A\}$ and the IP address of I back to the user U_A. Next, U_A will encrypt the security credentials of his cloud storage using key k_A and upload the ciphertext and R_A to the shared instance I. The security credentials will be decrypted in the execution of P, which takes R_A as an input parameter and apply the same hash function H on k_P and R_A. Multiple users could share the same instance and the server S will assign different random number R and key k.

$U_A \rightarrow S$: request a shared instance;
$S \rightarrow U_A$: I, R_A and k_A;
U_A encrypts security credentials SC_A with k_A, $\{SC_A\}_{k_A}$;
$U_A \rightarrow I$: R and $\{SC_A\}_{k_A}$.

File transfer between users: In Skyfiles, two users can also request a shared instance for transferring files between their cloud storages. Following the design in Section IV, the sender will initialize the process and request a shared instance from the server. Compared to the single user operations, file transfer requires both sender (U_A) and receiver (U_B) to send the security credentials of their cloud storage account to the shared instance. In addition, the sender needs to notify the receiver the instance assigned by S. The following Algorithm 2 shows the major messages exchanged in our design. We assume that the server holds a pair of public key/private key, indicated by k_S^+/k_S^- and the public key is known by U_A and U_B.

Algorithm 2 File Transfer with a Shared Instance

1: $U_A \rightarrow S$: request a shared instance
2: $S \rightarrow U_A$: I, R_A, k_A, and $\{U_A, I\}_{k_S^-}$
3: U_A encrypts security credentials with k_A, $\{SC_A\}_{k_A}$
4: $U_A \rightarrow I$: R_A, $\{SC_A\}_{k_A}$, F_{src}, and URI_F
5: I downloads F from F_{src} and stores it at URI_F
6: $U_A \rightarrow U_B$: I, URI_F, and $\{U_A, I\}_{k_S^-}$
7: $U_B \rightarrow S$: U_A and I
8: $S \rightarrow U_B$: R_B and k_B
9: U_B encrypts security credentials with k_B, $\{SC_B\}_{k_B}$
10: $U_B \rightarrow I$: R_B, $\{SC_B\}_{k_B}$, URI_F, and F_{dst}
11: I uploads F from URI_F to F_{dst}

Sender U_A initializes the request by contacting the server S. In the response, besides the same information for single user operations, the server sends an additional certificate back $\{U_A, I\}_{k_S^-}$, which is a signature of the requesting user's ID and the assigned instance I. U_A will upload the encrypted security credentials to I as well as the source file location (F_{src}) and intermediate file location (URI_F). Then U_A will notify the receiver U_B the shared instance I and the location of the target files (URI_F). This message is attached with the certificate from S so that the receiver can verify the shared instance I is legitimate. Next, the receiver U_B sends the server a request with (U_A, I). After verifying there exists a shared instance I serving U_A, the server will send back R_B and k_B to U_B so that U_B can encrypt his security credentials in the same way as U_A. Eventually, U_B uploads the encrypted security credentials and the intermediate file location (URI_F) and destination file location (F_{dst}) to the shared instance I.

VI. Performance Evaluation

In this section, we present the evaluation results of Skyfiles based on experiments. We have implemented Skyfiles system on Android with Dropbox [2] storage service and tested it on Google Nexus smartphone. For cloud-assisted operations, we use the service provided by Amazon Web Service [7] and all the experiments are conducted on the Micro instance. The major performance metrics we consider are time overhead and bandwidth consumption. For each particular setting, we conduct five independent experiments and the average values are reported in this section.

A. Basic file operations

We first present the bandwidth consumption of basic file operations implemented by official Dropbox APIs. In this test, we create a new Dropbox account with a folder "test" containing 1000 text files (22 bytes each). The operations we have tested are 1. log into dropbox; 2. create/delete a folder (under the root directory); 3. create/delete/rename a file (under "test"); 4. enter/leave a folder ("test").

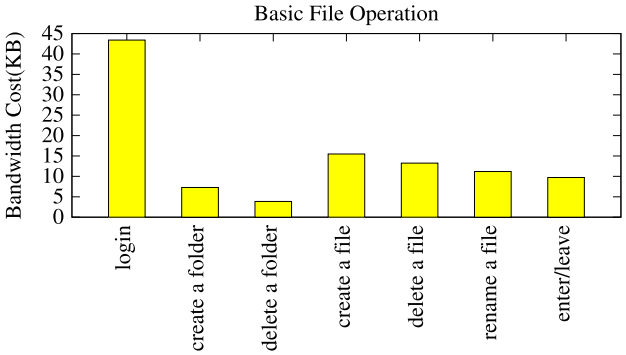


Fig. 3: Bandwidth Consumption of Basic File Operations

For each file operation, Dropbox server requires security credential to be attached and the communication is based on SSL. As shown in Fig. 3, the login process consumes the most bandwidth because of the interaction for authentication and Dropbox APIs that recursively fetch meta data to synchronize/update the local shadow file system. Creating and deleting an empty folder consumes 7.3K and 3.9K bytes respectively which are the minimum costs among the tested operations. Creating and deleting a text file is similar to the previous case. When tested in the folder "test," it certainly incurs more bandwidth cost (15.5K and 11.2K bytes). The reason is that the folder contains 1000 other files and once a change is made in the folder, Dropbox APIs will re-fetch the list of files in it. Finally, when a user enters the folder and then leaves, the bandwidth cost (9.7K bytes) is slightly lower than creating/deleting a file.

B. Cloud-assisted advance file operations

In this subsection, we evaluate the cloud-assisted file operations in Skyfiles, particularly download and compression. Due to the page limit, we omit the results for encrypt and convert operations. We first present the overhead of starting a cloud instance and then show the performance of these operations under the assumption that a cloud instance has been available. The workload we use for evaluation includes 4 sets of files: one picture (16M bytes), five pictures (83M bytes), and two video clips (63M and 127M bytes).

Overhead of starting a cloud instance: We conduct five groups of tests in this experiment at different time of a day. Each group contains 5 individual operation of starting a AWS Micro instance. The operation ends when the user is able to log into the instance. The following Fig. 4 illustrates the results of the average value and variance. Overall it is a time-consuming process as all the tested cases spend more than 15 seconds in starting an instance. After sending the request, the user has to

wait a long time until the cloud instance to be ready. In the rest part of this subsection, the performance overhead does not include the initial phase of starting an instance, thus it is for the case of using shared instance. If the user starts his own cloud instance, the extra overhead could range from 15 seconds to 30 seconds based on Fig. 4.

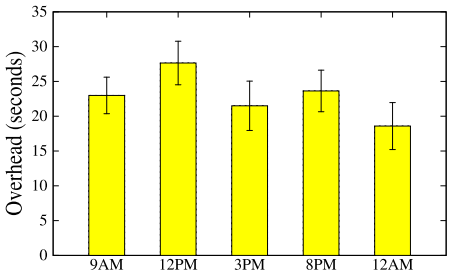


Fig. 4: Overhead of Starting a Cloud Instance

Overhead of download operations: In this test, we let the cloud instance download the files in our workload and upload them to our Dropbox storage space. The target files are hosted in one of our servers. As we can see from Table I, upload/download overhead is roughly proportional to the file size. Uploading is faster than downloading because the instance we use (AWS EC2) and the Dropbox service (AWS S3) belong to the same cloud service provider. Overall, the transmitting rate is 12.9Mbps which is much faster than downloading files to the smartphone and then uploading them to Dropbox cloud storage via cellular networks.

	1-Pic(16M)	5-Pics(83M)	Video1	Video2
Download time	14.7s	59.4s	34.2s	58.9s
Upload time	6.9s	31.8s	33.9s	58.7s

TABLE I: Overhead of Download Operations

Overhead of compress operations: In this experiment, we test compression operations on Dropbox files. Particularly, we use gzip to compress the files downloaded to the cloud instance and then upload the compressed file back to Dropbox cloud storage. Table II shows the breakdown time overhead of this operation. Uploading process normally costs the most, followed by downloading and compression process. This operation in Skyfiles is fast. For example, compressing one picture (16M) and 5 pictures (83M) cost 10.4s and 38.7s in total. The compressed files in these two cases are 7.7M and 40.0M bytes.

	1-Pic(16M)	5-Pics(83M)	Video1	Video2
Download time	4.9s	14.8s	19.1s	44.6s
Compression	0.8s	4.6s	4.2s	9.2s
Upload time	4.6s	17.9s	45.8s	70.4s

TABLE II: Overhead of Compress Operations

Bandwidth consumption of advance file operations: The bandwidth consumptions of advance file operations are similar as only control messages are being exchanged. We only show the performance of compress operation in Table III due to the page limit. Uplink cost which includes control messages from the smartphone to the cloud instance is very small. The downlink bandwidth varies for different file sizes and most of it is consumed by our periodical heartbeat messages reporting the status of the operation.

	1-Pic(16M)	5-Pics(83M)	Video1	Video2
Downlink cost	48.2K	147.4s	127.1K	184.0K
Uplink cost	3.9K	4.0K	3.8K	3.3K

TABLE III: Bandwidth Consumption of Compress Operations

C. File Transfer between Users

In this operation, the target files are downloaded by an instance from sender's storage and then uploaded to receiver's storage. When using shared instances, the extra costs for communicating with the trusted server is negligible compared to the total performance. The following Table IV lists the time overhead and bandwidth consumption of file transfer (both sender and receiver) with a shared instance. The binary program hosted by the server is 4.9M bytes in our implementation.

	1-Pic(16M)	5-Pics(83M)	Video1	Video2
Download time	8.5s	40.7s	31.4s	59.3s
Upload time	7.3s	25.8s	30.1s	59.8s
Bandwidth	63.4K	245.8K	221.2K	505.9K

TABLE IV: Performance of File Transfer (shared instances)

VII. Conclusion

This paper presents Skyfiles which uses cloud instances to help smartphone users manage files stored in cloud. Skyfiles includes an extended set of file operations without local file copies. In addition, Skyfiles supports file transfer between users and all the file operations in Skyfiles can be securely executed on shared instances.

References

[1] "iCloud," https://www.icloud.com/.
[2] "Dropbox," http://www.dropbox.com.
[3] "Box," http://www.box.com.
[4] "Google Drive," https://drive.google.com/.
[5] "Microsoft SkyDrive," https://skydrive.live.com/.
[6] "Ubuntu One," https://one.ubuntu.com/.
[7] "Amazon Web Service," http://aws.amazon.com/.
[8] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010.
[9] R. Kemp, N. Palmer, T. Kielmann, and H. E. Bal, "Cuckoo: A computation offloading framework for smartphones," in *MobiCASE*, 2010.
[10] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *INFOCOM*, 2012.
[11] A. Saarinen, M. Siekkinen, Y. Xiao, J. K. Nurminen, M. Kemppainen, and P. Hui, "Smartdiet: offloading popular apps to save energy," in *SIGCOMM*, 2012, pp. 297–298.
[12] I. Drago, M. Mellia, M. M. Munafo, A. Sperotto, R. Sadre, and A. Pras, "Inside dropbox: understanding personal cloud storage services," in *Proceedings of the 2012 ACM conference on Internet measurement conference*, ser. IMC '12, 2012.
[13] H. Wang, R. Shea, F. Wang, and J. Liu, "On the impact of virtualization on dropbox-like cloud file storage/synchronization services," in *Proceedings of the 2012 IEEE 20th International Workshop on Quality of Service*, ser. IWQoS '12, 2012.
[14] M. V. Barbera, S. Kosta, J. Stefa, P. Hui, and A. Mei, "Cloudshield: Efficient anti-malware smartphone patching with a p2p network on the cloud," in *P2P*, 2012, pp. 50–56.
[15] S. Fahl, M. Harbach, T. Muders, and M. Smith, "Confidentiality as a service – usable security for the cloud," in *Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, 2012.
[16] "OAuth v1.0," http://oauth.net/core/1.0a/.
[17] "Microsoft Azure," http://www.windowsazure.com.
[18] "HP Cloud," https://www.hpcloud.com.