Computer Networks 103 (2016) 228–239

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Characterizing mobile *-box applications

Xiapu Luo[a,c], Haocheng Zhou[b], Le Yu[a], Lei Xue[a], Yi Xie[b,d,*]

[a] Department of Computing, The Hong Kong Polytechnic University, HK SAR, China
[b] Fujian Key Laboratory of Sensing and Computing for Smart City, School of Information Science and Engineering, Xiamen University, China
[c] The Hong Kong Polytechnic University Shenzhen Research Institute, China
[d] Xiamen University ShenZhen Research Institute, China

ARTICLE INFO

Article history:
Received 17 October 2015
Revised 21 February 2016
Accepted 31 March 2016
Available online 25 April 2016

Keywords:
Personal cloud storage service
Mobile applications
Mobile and personal devices
Performance

ABSTRACT

With the increasing use of multiple electronic devices including tablets, PCs, and mobile devices, Personal Cloud Storage (PCS) services, such as Dropbox and Box, have gained huge popularity. Recent research has used the PC clients of a few PCS services to study the network architectures and performance of these services. The mobile clients deserve a further study because the study of PC clients does not necessarily represent the system and network demand with mobile clients. In this paper, we conduct the first systematic investigation on six popular PCS services to reveal their internals and measure their performance. By dissecting their protocols and conducting cross-layer examinations, we obtain interesting observations, identify design issues, and suggest solutions to remedy these issues. Moreover, we propose an efficient method to measure the response latency of PCS servers by exploiting their open APIs.

1. Introduction

Personal Cloud Storage (PCS) services have become very popular because they support file synchronization and sharing through multiple devices and platforms. Bundling PCS with smartphones makes PCS even more accessible to a large number of users. For example, Dropbox's and Google drive's Android clients have 1-5 *billion* installations [16]. Moreover, Samsung supplies free 2-year 50GB Dropbox space to Galaxy users [35] and HTC users are provided 2-year 50GB of free Google drive space [20].

Although recent research has studied the network architectures and performance of a few PCS services using their PC clients [8,9,21,27,39], little is known about their mobile clients. The designs of mobile clients (or mobile applications, apps) are deserving of careful study because of the differences between PC and smartphone [22,32,33]. More precisely, compared with PC, smartphones generally have less powerful CPU, less memory, and limited battery lifetime. Furthermore, mobile communication usually has worse performance than wired communication, in terms of delay/jitter, loss rate, and instable connectivity. Moreover, mobile data is much more expensive than wired data. Therefore, it is desirable for apps to achieve a good balance between performance and resource consumptions including energy and bandwidth.

In this paper, we fill these gaps by conducting the first systematic examination on six popular PCS services. We characterize their mobile clients (*-box) from three aspects (i.e., protocol, client, and server), and compare them with their PC clients. First, instead of treating them as black boxes, we dissect their protocols to understand their behaviors. It is non-trivial to achieve this purpose because they usually use HTTPs to encrypt their traffic and even adopt SSL pinning [12] to prevent the man-in-the-middle (MITM) attacks. Although Drago et al. modified Dropbox's trusted certificate to conduct MITM attacks for reverse engineering Dropbox's protocol [9], such approach may not be always feasible, because a client can just check some selected fields in a certificate without loading the embedded certificate into memory. We tackle this problem by proposing a more general hooking-based method to analyze PCS clients that are resistant to MITM attacks. The recovered protocols provide us detailed information to explain interesting observations and uncover design issues.

Second, we examine the functionality and features of mobile clients, especially those that will affect performance and energy consumption. By carefully designing experiments as well as correlating information from different sources (i.e. the traffic/performance analysis in PC clients and mobile clients), we identify several design issues in these PCS mobile clients, which may lead to low throughput and high energy consumption. Finally, we profile the servers of these PCS services, and develop SyncTest, a tool for measuring the response latency of a PCS server by exploiting open APIs.

In summary, our major contributions include:

* Corresponding author.
E-mail address: csyxie@xmu.edu.cn (Y. Xie).

http://dx.doi.org/10.1016/j.comnet.2016.03.025
1389-1286/© 2016 Elsevier B.V. All rights reserved.

Table 1
PCS Services under measurement and their properties. Note that Dropbox's mobile client activates file deduplication after the file size exceeds a threshold.

PCS	Interface	Mobile client				PC client			
		Compress	De-duplicate	Bundle	Heart-beat(Sec)	Compress	De-duplicate	Bundle	Heart-beat(Sec)
Baidu	RESTful	NO	YES	NO	0	NO	YES	NO	5
Box	RESTful	NO	NO	NO	476	NO	NO	NO	488/608
Dbank	API	NO	YES	NO	0	NO	YES	NO	171
Dropbox	API	NO	YES	NO	0	YES	YES	YES	60
Gdrive	API	NO	NO	NO	0	YES	NO	NO	20
Onedrive	RESTful	NO	NO	NO	0	NO	NO	NO	40

- To our best knowledge, we conduct the first systematic measurement on six popular PCS services.
- We build a measurement system to automatically drive mobile clients and collect traces for analysis. To scrutinize the protocols protected by HTTPS, we propose an inline hooking based approach to dump the protocols' content.
- We design experiments to characterize the functionality and features of mobile clients and PC clients, and confirm them by correlating information from different sources. We further examine the similarities/differences between their mobile clients and PC clients. Moreover, besides having new observations, we identify design issues in their clients and open APIs, shed light on their impacts, and suggest solutions to remedy them.
- By exploiting open APIs, we design and implement SyncTest for measuring the response latency of PCS servers. It is the first work to characterize PCS services' response latency.

The rest of this paper is organized as follows. Section 2 describes our measurement methodology and introduces the PCS services under investigation. Section 3 details our findings from six mobile PCS clients, and Section 4 enumerates our suggestions to improve mobile clients. After introducing the related work in Section 5, we conclude the paper with future work in Section 6.

2. Mobile PCS clients and measurement methodology

2.1. Personal cloud storage services

Table 1 enumerates the PCS services under measurement and their properties, some of which are elaborated in Section 3. Box [4], Dropbox [11], Google Drive (Gdrive) [15], and Microsoft's OneDrive [29] are global popular PCS services. Baidu [2] and Dbank [23] are two major PCS services in China. The mobile clients' and PC clients' version numbers are listed in Table A.10 of Appendix A. Since Android has occupied 81.5% market share with millions of apps, we examine the PCS services' Android clients (or Android apps). It is worth noting that our methodology could be applied to mobile clients on other platforms. To foster the usage, some PCS services offer proprietary APIs, while others supply Representational State Transfer (RESTful) [13] APIs through which clients can manipulate files using standard HTTP requests.

2.2. Measurement methodology

We characterize each PCS service from three aspects.

Protocol. We dissect the protocol of each PCS service to understand the interactions between the server and the client, and examine its contents, such as HTTP headers. This investigation allows us to explain interesting observations and identify design issues (e.g., chunking, heartbeat, etc.). For example, we have a new observation from the content of Dropbox's protocol. The field 'x-server-response-time' records the response latency of Dropbox server, which has not been shown in the existing studies [8,9].

Client. We design experiments to characterize PCS clients, including the functions that can save bandwidth and energy (e.g., compression, deduplication, bundling, transmission resuming at break-points, etc.), and the features that can affect performance and energy consumption (e.g., chunking, packet size, TLS records size, and periodic transfer). Moreover, we set up an environment to measure the energy consumption of mobile PCS clients.

Server. The information of PCS servers (e.g., IP address, geolocation etc.), which provides some hints to the service performance, is collected in two ways. One is to extract IP addresses from the traces of traffic sent/received by mobile clients. The other is to run SyncTest on 10 planetlab nodes in 6 countries for collecting host names, and then resolve their IP addresses.

2.3. Measurement system

The measurement system includes: (1) a module for automatically uploading/downloading files; (2) a module for extracting contents from HTTPs traffic; (3) a module for collecting information from different sources; and (4) SyncTest for measuring servers' response latency.

Upload/download files.

Since PC clients automatically check whether a local folder contains the same files as the corresponding folder in the server, we prepare a set of python scripts to add or remove files from a local folder to trigger the file uploading/downloading. In contrast, mobile clients are User Interface (UI) centric and do *not* have this function. Therefore, we use UI Automator [17], a library from Google for conducting UI test, to automate the uploading/downloading process of selected files.

Extract the contents from HTTPs traffic.

We adopt two approaches to obtain HTTPs traffic and extract its contents.

The first one is to launch MITM attacks on PCS clients using *fiddler* [38]. More precisely, we insert the fiddler's root certificate into OS's trusted root certificate store and then redirect the traffic to *fiddler*. Surprisingly, most clients, especially Android clients, are vulnerable to this attack, although this issue has been raised many times [12]. Note that only the PC clients of Box, Dropbox, and Gdrive adopt the certificate pinning [12] that compares the received certificate with the embedded one to defend against MITM attacks.

The second one is to propose an inline hooking based method for PCS clients that use certificate pinning to defend against MITM attacks. We propose a general method that dumps the outgoing traffic contents *before* encryption and saves the incoming traffic contents *after* decryption. More precisely, we first inject our program in dynamic link library (dll) into a target client. In order to dump outgoing requests, our program modifies a few bytes at the beginning of the function *SSLSend* so that the execution will be redirected to our dumping function before the traffic contents are encrypted. After that, the dumping function saves the contents and then resumes the execution in *SSLSend*.

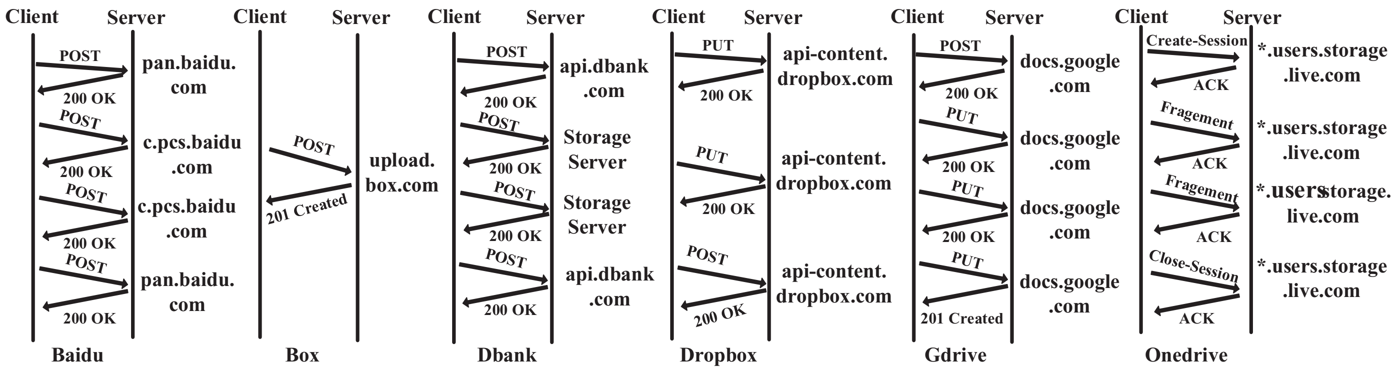


Fig. 1. The interactions between mobile clients and PCS servers during file upload.

Collect the measurement data.

We capture the incoming/outgoing traffic at the client side using TCPDump and record the states of TCP (e.g., congestion window, etc.) using TCP_m_Probe [6]. In order to obtain the geolocation of each IP, we use Maxmind's GeoLite databases [28] and check the IP's AS number using the mapping from IP to ASN [37].

SyncTest.

SyncTest manipulates files (e.g., file uploading, file downloading, and file deletion) by invoking the proprietary APIs or REST APIs of PCS services. Before conducting the measurement, SyncTest obtains an access token through the OAuth protocol [5]. For Baidu, Box, Dbank and Onedrive, we get the access token by sending HTTP requests to their remote servers. For Dropbox and Gdrive, we get the access token by calling their APIs [1]. After obtaining an access token of the PCS service, SyncTest conducts various file operations through REST interfaces or propriety APIs.

Then, we leverage the RESTful interfaces of Baidu, Box, and Onedrive to execute different actions [1,3,31]. The RESTful architecture utilizes URIs (uniform resource identifier) to identify web resources and reuses HTTP methods (e.g., GET, POST, DELETE, etc.) to manipulate web resources [34]. SyncTest sends HTTP requests with different HTTP methods to upload or download or delete files. For example, the following HTTP request is issued to upload a file named "Helloworld" to Onedrive.

"PUT https://apis.live.net/v5.0/me/skydrive/files/HelloWorld.txt?access_token=ACCESS_TOKEN ".

To download or delete a file, we use the following HTTP requests, respectively.

"GET https://apis.live.net/v5.0/FileID/content?access_token=ACCESS_TOKEN ".

"DELETE https://apis.live.net/v5.0/FileID?access_token=ACCESS_TOKEN "

We utilize the propriety APIs of Dbank, Dropbox, and Gdrive to execute file operations [7,10,14]. For example, the method `DbxClient.uploadFile()` is used to upload a file to Dropbox. To download or delete a file, we invoke `DbxClient.getFile()` and `DbxClient.delete()`, respectively.

SyncTest can also measure a PCS server's response delay $T_{response}$ (i.e., the time used to deal with a request). As shown in Fig. 6(a), SyncTest records $T_{measure}$, which is the period between sending a request and receiving the *first* TCP data packet from the server. More precisely, SyncTest starts TCPDump and records the time of sending a request, denoted as T_{ms}. From the packets captured by TCPDump, SyncTest can identify the *first* response data packet and extract its arriving time, denoted as T_{me}. Then, $T_{measure} = T_{me} - T_{ms}$.

Obviously, $T_{measure}$ contains a round-trip time (RTT) and the server's response latency ($T_{response}$). Then we approximate $T_{response}$ by $T_{measure} - \min\{RTT\}$, where the RTT samples are obtained by analyzing the traces [24]. Since RTT may be affected by many factors, to reduce the impact of RTT on $T_{response}$, we could deploy SyncTest in the hosts that are close to the target PCS servers with stable RTTs. Alternatively, if the PCS service uses content distribution network (i.e., CDN), we could measure the PCS servers that are near to the measurement hosts.

3. Measurement results

3.1. Protocol

File uploading.

Fig. 1 illustrates the interactions between a mobile client and its servers during file uploading for each PCS service.

Box adopts a simple approach. After a user selects a file and presses the upload button, the mobile client uploads the file using the HTTP POST method. After receiving the whole file, the server replies with an HTTP message '201 Created' as well as the file's metadata (e.g., name, id, owner, path, SHA1) in the JSON format.

Dropbox, Gdrive and Onedrive use one TCP connection for file uploading and only one server is involved in the process. They differ in how they split the file before transmission (i.e., chunking) and how to convey the fragment information. Both Dropbox and Gdrive use the HTTP PUT method to upload fragments. Dropbox records the fragment size in the HTTP request header with the *Content-Length* field. And its server responds with '200 OK' as well as the fragment's hash value in payload.

Gdrive uses the *Content-Length* and *Content-Range* fields to indicate the file size and the chunk size. Its server responds '201 Created' or '308 Resume Incomplete' [2] depending on whether the current chunk is the last one. Moreover, the response header contains HTTP Range option showing the amount of bytes received by Gdrive's server.

Onedrive uses the Background Intelligent Transfer Service (BITS) protocol [3] to upload the selected file, which was designed for communication with frequent disconnections. Its mobile client firstly sends a CREATE-SESSION message to create a BITS session. Then, the client uploads the file in a series of FRAGMENT messages, and the server replies with ACKs to confirm the received data by bytes. The session will be closed after all messages are sent.

Baidu and Dbank adopt a two-level network architecture. They use one server to handle the upload request and the other server to receive the file. More precisely, after dividing a file into several

[1] The API of access token for Dropbox is `WebAuthSession.getAuthInfo()`, while the one for Gdrive is `GoogleCredential.getAccessToken()`.

[2] https://code.google.com/p/gears/wiki/ResumableHttpRequestsProposal

[3] http://msdn.microsoft.com/en-us/library/cc216518.aspx

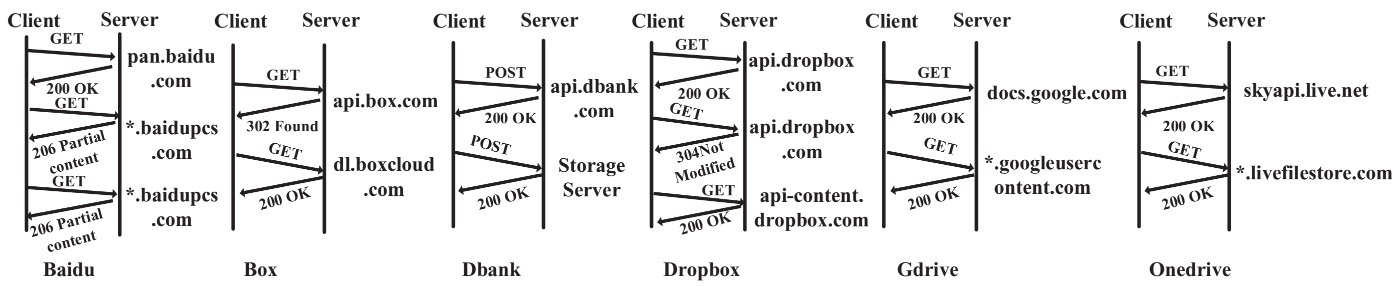


Fig. 2. The interactions between mobile clients and their servers during file download.

blocks, the mobile client of Baidu firstly posts the information of blocks (e.g., file name, size, directory, each block's MD5, device id, etc.) to the server 'pan.baidu.com', which replies with '200 OK' as well as the file name and a block list. Then, the client uploads the blocks one by one to the server 'c.pcs.baidu.com', which replies with '200 OK' as well as the MD5 value of each block. Similarly, the mobile client of Dbank sends the information of file blocks to the server 'api.dbank.com', and obtains the IP address and keys of a storage server. After that, the client uploads all blocks to the storage server.

By analyzing the interactions between PC clients and servers, we found that all except Box have the same pattern as those shown in Fig. 1. Instead, Boxs PC client first contacts 'api.box.com' using the HTTP OPTIONS method, and sends the files name and size to this server. The server will send back a URL for uploading the file and the client will post the file to this URL.

File downloading.

Fig. 2 illustrates the interactions between the mobile clients and the servers of six different PCS services during file downloading. All interactions follow a similar pattern. That is, the client sends the information of the file to a PCS server, which will redirect the client to *another* server for file downloading. For example, Dbank's server (i.e., 'api.dbank.com') sends the IP address of the storage server to the client. Other PCS servers will reply with the host name of the downloading server for file operation.

Dropbox's and Onedrive's PC clients have different behaviors from their mobile clients. More precisely, the mobile clients send an HTTP request and then download the whole file as shown in Fig. 2. If the TCP connection is broken, their mobile clients have to download the whole file. In contrast, their PC clients use chunking during file downloading, where each HTTP request only fetches a block of data. Therefore, when the TCP connection fails, the PC clients know how to continue the downloading.

3.2. Client

Capability.

Table 1 compares the capability of PC clients and mobile clients of six PCS services in terms of compression, deduplication and bundling. With the support of these three properties, clients could communicate efficiently with the servers while consuming less energy. For example, the PC client of Dropbox supports all properties, while its mobile client only supports deduplication when the file size is larger than a threshold. Baidu and Dbank provide deduplication in both their PC and mobile clients. We conduct file uploading and downloading to identify various capabilities, and the basic steps of file uploading and downloading are listed in Appendix B.

File uploading.

We conduct three types of file uploading to examine PCS clients' features related to incompressible file uploading, compressible file uploading, and bundled file uploading.

Table 2 records the volume of traffic involved in the incompressible file uploading through different clients. It shows that

Table 2
The volume of traffic involved in basic file uploading (unit: Byte).

PCS	Android client			PC client		
	100K	1M	10M	100K	1M	10M
Baidu	128K	1.08M	10.7M	126K	1.16M	11.6M
Box	189K	1.22M	11.2M	134K	1.21M	11.3M
Dbank	147K	1.09M	10.6M	136K	1.18M	11.7M
Dropbox	129K	1.13M	11.1M	155K	1.30M	13.0M
Gdrive	148K	1.15M	11.2M	141K	1.14M	11.1M
Onedrive	157K	1.23M	12.3M	124K	1.20M	11.0M

Table 3
The volume of traffic involved in uploading compressible files (unit: Byte).

PCS	Android client			PC client		
	100K	1M	10M	100K	1M	10M
Baidu	130K	1.09M	11.2M	130K	1.17M	11.1M
Box	208K	1.26M	11.2M	141K	1.13M	11.1M
Dbank	146K	1.10M	10.7M	134K	1.20M	11.0M
Dropbox	129K	1.13M	11.0M	**76K**	**535K**	**5.7M**
Gdrive	121K	1.12M	11.1M	**62K**	**502K**	**4.9M**
Onedrive	136K	1.25M	12.2M	141K	1.24M	11.4M

when sending small files the additional overhead accounts for a large percent of the traffic. For example, to upload a 100KB file, Box's mobile client generates 89% additional traffic (i.e., (189K-100K)/100K = 0.89). In contrast, when uploading a 10MB file, it only produces 12% additional traffic (i.e., (11.2M-10M)/10M = 0.12). Therefore, avoiding the transmission of individual small files can improve the efficiency. One possible approach is to use file bundling, which combines several small files into a large one before uploading them.

To evaluate the compressible file uploading, we first generate compressible text files using an English dictionary, and then compare the file size with the amount of traffic captured during the file uploading. As shown in Table 3, only the PC clients of Dropbox and Gdrive compress files before file uploading, because the traffic volume (highlighted in red) is smaller than the original file size. It is worth noting that none of the mobile clients conducts compression before file uploading, because the amount of traffic is always larger than the original file size.

While [25] observed that the PC client of Gdrive does not perform compression before file uploading, we conduct the MITM attack to inspect the traffic to/from Gdrive's PC client, and confirm that it performs compression before file uploading. More precisely, its PC client firstly splits a large compressible file into small blocks with size of 8MB, and then uploads the compressed blocks one by one. For example, a 21MB file is split into three blocks (i.e. 8MB, 8MB, and 5MB), where each block is compressed into a file less than 3MB. Three compressed blocks are then uploaded to Gdrive, and the total traffic amount is less than 9MB, which is much smaller than 20MB, the original file size.

Table 4
The volume of traffic involved in file uploading test using Dropbox's mobile client(unit: Byte).

File Size	2M	3M	4M	5M	6M	7M	8M	9M
Traffic	2.23M	3.39M	4.61M	5.76M	6.92M	7.85M	9.04M	10.07M

Table 5
The volume of traffic involved in uploading 100 10KB files. (unit: Byte).

PCS	Android client	PC client
Baidu	1.21M	1.22M
Box	1.60M	1.56M
Dbank	4.14M	1.21M
Dropbox	1.21M	1.30M
Gdrive	2.12M	2.20M
Onedrive	1.62M	1.83M

Table 6
The volume of traffic involved in duplicate file uploading (unit: Byte).

PCS	Android client			PC client		
	100K	1M	10M	100K	1M	10M
Baidu	**13K**	**17K**	**16K**	**14K**	**12K**	**15K**
Box	187K	1.20M	11.4M	131K	1.11M	10.9M
Dbank	**33K**	**28K**	**30K**	**16K**	**17K**	**16K**
Dropbox	128K	1.04M	**15K**	**26K**	**25K**	**26K**
Gdrive	122K	1.12M	11.2M	122K	1.10M	10.9M
Onedrive	126K	1.20M	12.1M	122K	1.10M	11.0M

Moreover, while [25] observed that the mobile client of Dropbox would compress file and result in less traffic, we did not see it according to our experiments. As shown in Table 4, Dropbox's mobile client does not employ any compression, because the amount of traffic is always larger than the original file size. To further confirm our finding, we have analyzed the file uploading procedure of Dropbox's mobile client by conducting the MITM attack. We find that Dropbox's mobile client splits a text file into blocks with size of 4MB before file uploading, and then puts each block directly into an HTTP message *without* performing any compression.

To check whether PCS clients support file bundling (i.e., concatenate several small files into a larger one before file uploading), we instruct both mobile clients and PC clients to upload 100 files together, each of which is of 10KB. Table 5 lists the volume of traffic involved in this experiment. By comparing the results in Table 5 and the results of uploading one 1MB file in Table 2, we find that since mobile clients do not support file bundling, the total amount of traffic involved in uploading 100 10KB files is larger than that of uploading one 1MB file. Moreover, by analyzing the traffic, we have several interesting observations. First, Dropbox's mobile client and PC client use the least traffic among the clients using HTTPs (i.e., Dropbox, Box, Gdrive, and Onedrive). The reason may be that Dropbox only uses one HTTPs connection to send the files while the mobile clients of Box, Gdrive, and Onedrive use 2-4 HTTPs connections to transmit these files. The PC clients of Box and Onedrive use 6-7 HTTPs connections to transmit these files while Gdrive's PC client creates one HTTPs connection for each file.

Both mobile and PC clients of Baidu use 4 TCP connections to upload these files. In contrast, Dbank's mobile and PC clients create one TCP connection to upload one file. It is worth noting that Dbank's mobile client results in much more traffic because it queries 'api.dbank.com' to get a list of files after uploading a file. The file list is not small for 100 files (highlighted in red in Table 5). Dbank's PC client does not have such behavior.

File deduplication.

File deduplication is an important function for PCS services, which efficiently avoids transmitting duplicate files. In this experiment, we first upload a file to one PCS service, and then re-upload this file after changing the filename. If the PCS service implements deduplication, the traffic captured in the second file uploading will be much less than the original file size.

As shown in Table 6, Baidu, Dbank and Dropbox have implemented the deduplication function in their PC and mobile clients, because that the second file uploading only creates 15-30KB traffic (highlighted in red), which is much less than the original file size. But there is an exception. The mobile client of Dropbox will upload the duplicate file when the file size is small, and it only activates file deduplication when the file size is large (i.e. 10M file). To further study the rule of Dropbox, we have carried out many experiments of duplicate files with different file sizes. As shown in Table 7, the approximate threshold of Dropbox's deduplication in terms of file size is between 7.5M and 8M (highlighted in red).

Chunking.

PCS clients usually split a large file into several pieces and then upload them according to different strategies. Fig. 3(a) lists the strategies of six PCS services. '-' means that the file is transmitted as a whole. 'O' denotes that all chunks are transmitted in one TCP connection. 'I' indicates that each chunk is transmitted in a newly established connection. 'P' means that multiple chunks are transmitted in parallel connections. 'R' shows that transmission resuming is supported at break-points. 'Y/N' indicates whether or not an API is provided to customize the size of chunk. Some interesting findings are shown as below.

Chunking may ease error recovery and facilitate delta encoding[8,9]. Since mobile connection is usually instable (e.g., frequent disconnection), we examine whether PCS clients support transmission resuming at break-points, which can save energy and bandwidth. Fig. 3(a) shows that all PCS clients except Gdrive's support this feature when they adopt chunking. Gdrive's PC client supports chunking, but it will download the whole file again if the connection is broken. Since Box does not employ chunking, its client will upload or download the whole file when the connection is resumed. Although DBank's client does not use chunking for downloading files, it can resume the downloading at the break-points by using HTTP Range option. Similarity, Gdrive's Android client does not employ any chunking strategy and uses HTTP Range option to continue the downloading at the break-points.

It is tricky to decide the chunk size. Fig. 3(a) shows that PCS clients may adopt diverse values in different scenarios. Baidu and Dropbox selects 4M while Dbank uses 0.25M for both mobile and PC clients. The values in GDrive's and Onedrive's mobile clients are smaller than those in their PC clients.

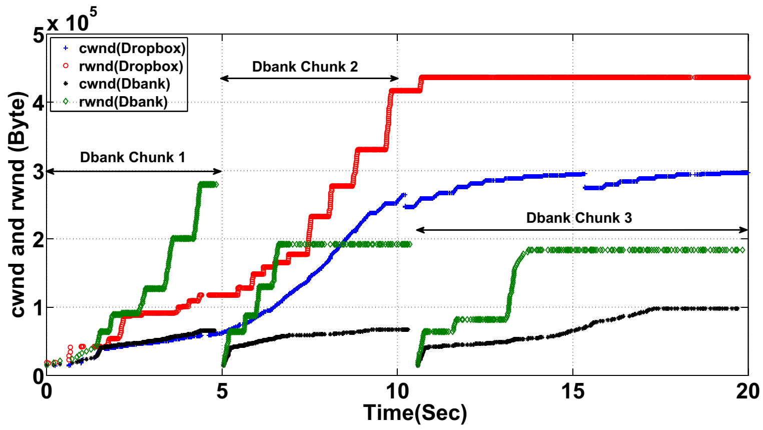
PCS clients may use one or multiple TCP connections to transmit these chunks. When uploading a file, some clients uses one TCP connection to send all chunks. However, Baidu's and Dbank's mobile clients create a new TCP connection to upload each chunk. Note that establishing a new TCP connection expands the uploading duration. Furthermore, selecting a small chunk size may also lead to low throughput because the congestion window (cwnd) cannot reach a large value to make full use of the capacity. Fig. 3(b) shows the cwnd and the advertised window (rwnd) of Dropbox's and Dbank's mobile clients when a large file is being uploaded. Since Dbank uses a small chunk size (0.5M), its cwnd is still small after sending one chunk. In contrast, since Dropbox adopts a large chunk size, its cwnd can almost reach the capacity of the mobile channel, thus having a higher throughput.

Table 7
The volume of traffic involved in Dropbox's duplicate file uploading with different file sizes. (unit: Byte).

File Size	3M	4M	5M	6M	7M	7.5M	8M	9M
Traffic	3.48M	4.47M	5.65M	6.85M	7.81M	8.41M	**24K**	**19K**

PCS	Upload Chunk Size			Download Chunk Size		
	Android	PC	API	Android	PC	API
Baidu	4M|I|R	4M|O|R	Y	50M|I|R	-	N
Box	- |O	-|O	N	-|O	-	N
Dbank	0.5M|I|R	0.5M|I|R	N	-|O|R	-|R	N
Dropbox	4M|O|R	4M|P4|R	Y	-|O	4M|O|R	N
Gdrive	0.25M|O|R	8M|I|R	Y	-|O|R	8M|O	N
Onedrive	1M|O|R	4M|O|R	N	-|O	1M|O|R	N

(a) The size of chunk, the number of TCP connections used for chunk transmission, and the support of transmission resuming at break-points.



(b) The congestion window (cwnd) and advertised window (rwnd) of Dropbox/Dbank.

Fig. 3. The effects of the chunk size and the number of TCP connections.

Table 8
How are file changes in the server side synchronized to the mobile clients? (B: back button; H: home button).

PCS	Main activity	Activity change	Press refresh button	Server notification
Baidu	NO	NO	YES	NO
Box	YES(B)	NO	YES	YES
Dbank	YES(HB)	NO	YES	NO
Dropbox	YES(B)	YES	YES	NO
Gdrive	YES(B)	YES	YES	NO
Onedrive	YES(HB)	YES	YES	NO

The PC clients of Dropbox, Gdrive and Onedrive use chunking and the HTTP Range option for downloading files. They limit the response size to as 4M, 8M and 1M, respectively. For example, Dropbox first splits a file into several blocks and puts the hash values and lengths of these blocks into the HTTP payload. This data is further encoded through HTTP's chunked transfer encoding before being delivered to a client.

Synchronization between the PCS server and mobile client. One of the salient features of PCS is to automatically synchronize files between clients and servers. We study the synchronization operations between the PCS servers and their mobile clients from three aspects.

Firstly, we modify a file in a PCS server and check how the mobile client updates the local file. Table 8 shows that only Box's server notifies its client about the changes. Other PCSs' mobile clients query their servers for changes during certain GUI events, such as when the main activity gets focused, the activity transitions happen, and the refresh button is pressed.

Secondly, we download a file from a PCS server through its mobile client to the smartphone and modify the file through ADB (Android Debug Bridge).[4] We find that none of mobile clients will send the file changes to the server. Unlike PC clients, mobile clients do not provide the synchronization functionality. In contrast, if we just modify the file through the mobile client without downloading it (i.e., the file is still in the server), the changes will be immediately applied to the file in the server.

Thirdly, we download a file from a PCS server to the smartphone and modify the local file and the remote file respectively. When the file is downloaded again, the mobile clients of Baidu, DBank, Gdrive and Onedrive will save the downloaded file using a different file name. Box will replace the local file with the newly downloaded file while Dropbox will ask users whether the local file shall be replaced.

[4] https://developer.android.com/tools/help/adb.html

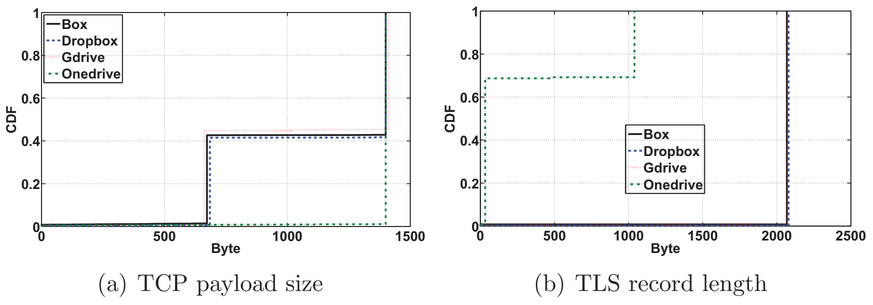


Fig. 4. CDF of TCP payload size and TLS record length.

Packet size vs. TLS record length.

Since sending packets consume much energy, mobile clients should improve the transmission efficiency. However, we find the mobile clients of Box, Dropbox and Gdrive transmit many small TCP segments which lead to inefficient transmission. As shown in Fig. 4(a), when a 20M file is uploaded to these PCS services, 40% TCP segments are around 750 bytes, which is much less than the maximal segment size (MSS).

The reason is revealed in Fig. 4(b), which depicts the distribution of the TLS record length. All PCS clients except Onedrive set the TLS record length to around 2070 bytes. Therefore, two TCP packets are needed to transmit one TLS record (i.e., one conveys MSS bytes and the other one contains the remaining content). By selecting a small TLS record length, Onedrive follows a good best practice by having each TCP packet carry one TLS record [18].

Periodic transmission (Heart beat detection).

All PC clients periodically connect to certain servers for multiple purposes, for example detecting the changes of files, accessing the report logs. The periods are different in these PCS services, as shown in the heart-beat intervals of Table 1. It is interesting that Box's PC client communicates with two servers with different periods, including the server '2.realtime.service.box.net' for 488s and the server 'client-log.box.com' for 608s. The former is used to support the long polling for getting real-time notifications of events.[5] Since periodic transmission may drain the battery [32], most mobile clients do not have this function. The exception is Box, whose mobile client periodically contacts the server '2.realtime.service.box.net'.

3.3. Server

IP and geolocation.

We use 10 Planetlab nodes around the world to study the storage servers of six PCS services, and measure the round-trip time (RTT) between each Planetlab node and the connected server. Fig. 5(a) illustrates the geo-locations of all Planetlab nodes and all storage servers of PCS services except Gdrive. The measured RTTs are shown in the boxplots of Fig. 5(b). Note that different Planetlab nodes may connect to different servers whose host names are the same. The server selection depends on the load balancing policy of PCS services. After carefully analyzing the traffic captured by Planetlab nodes, we have the following interesting findings.

Some PCS services use a few IPs to serve all clients. For example, Box uses 'upload.box.com' and 'dl.boxcloud.com' for uploading and downloading files, respectively. The former is resolved to two IP addresses (i.e., 74.112.184.71 and 74.112.185.71) and the latter is resolved to two IP addresses (i.e., 74.112.184.74 and 74.112.185.74). All servers belong to AS33011 and are located in Santa Clara, US.

Some PCS services use many servers to handle client's requests, but these servers are often deployed in one region. For example, the servers of Dropbox, Box, and Onedrive are located in US while the servers of Baidu and Dbank are located in China. More precisely, Dropbox uses 'api-content.dropbox.com' for uploading and download files, and 20 IP addresses are associated with this host name. All servers belong to AS14618 and locate in Virginia, US. It is consistent with the observation in [8]. Therefore, the clients far from these IPs would suffer from long RTT.

In contrast, Gdrive adopts the CDN technique to assign the nearby servers to clients. Its RTTs between the server and client are much smaller than that of other PCS services due to the usage of suitable servers, as shown in Fig. 5(b). More precisely, Gdrive uses 'docs.google.com' and '∗.googleusercontent.com' for file uploading and downloading, respectively. We find that 180 IP addresses are associated with the former and 60 IP addresses are associated with the latter. These IPs spread around the world.

Response latency.

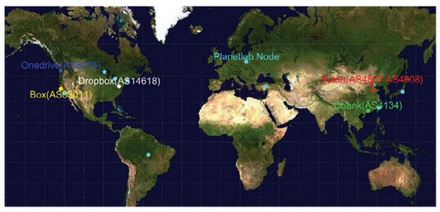
We find a field named 'x-server-response-time' in the response of Dropbox, and infer that this option records the response delay of Dropbox's server. In order to verify it, we redirect the traffic to fiddler for extracting its value and compute $T_{response}$ at the fiddler side. We carry out the experiments every 5 minutes for around 6 hours, and show the upload/download response delays estimated through 'x-server-response-time' and $T'_{response}$ in Fig. 6(a). It is obvious that the response delays from the field of 'x-server-response-time' (i.e., marked with *field*) well match the values of $T'_{response}$ (i.e., marked with *pcap*). Moreover, the response delays of the download operation are usually smaller than that of the upload operation. The reason may be that the upload operation will write data in the cloud and therefore require longer time [40].

We also measure the upload/download response delays of other PCS services. The experiment results are shown in Fig. 6(c). In general, download operations have shorter response delays than upload operations. Gdrive has the largest response delay and the largest variance of delay. Dbank has the lowest response delay because its API does not use TLS to encrypt the message.

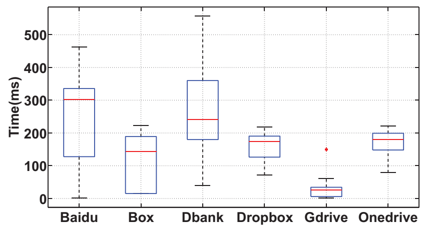
3.4. Energy consumption

Fig. 7 (a) illustrates the configuration for measuring energy consumption of mobile PCS clients. Laptop1 with Internet connection acts as a WiFi access point (AP) so that it can capture all traffic. DC electrical source (MPS-3003L) provides a stable voltage to the mobile client (Samsung smartphone GT-I9300) with PCS clients. The power meter can measure the voltage and current values of the smartphone. Laptop2 then obtains the measurement data from its

[5] https://developers.box.com/

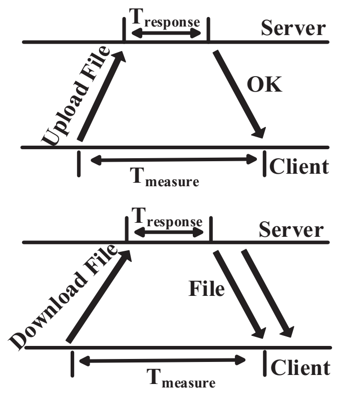


(a) Geo-location of Planetlab nodes and the PCS storage servers (except Gdrive's).

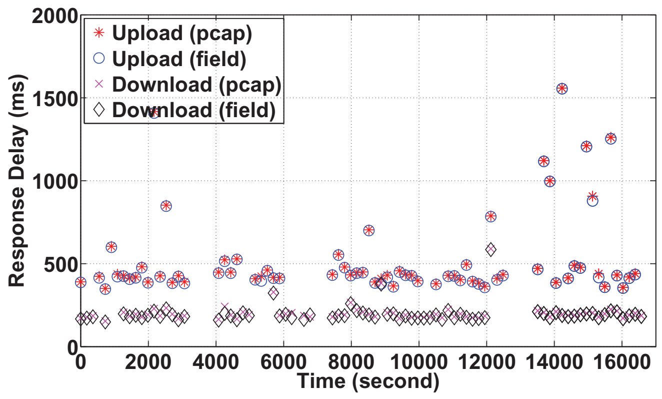


(b) Round-trip time between each Planetlab node and the server.

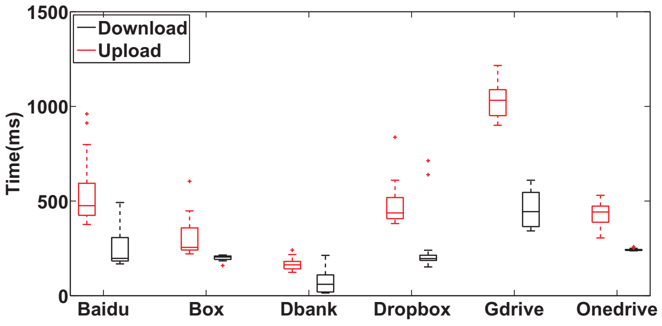
Fig. 5. Characterizing storage servers.



(a) Measurement response delay by exploiting APIs in the fiddler.



(b) Response delay of Dropbox in upload/download operations.



(c) Response delay of PCS servers in upload/download operations.

Fig. 6. Measuring the response delay of PCS servers.

Table 9
Energy consumption in uploading/downloading compressible files.

PCS	Upload			Download		
	100K	1M	10M	100K	1M	10M
Baidu	2.1J	5.8J	15.5J	2.5J	6.7J	17.9J
Box	3.2J	9.4J	29.9J	2.7J	9.4J	26.6J
Dbank	3.4J	4.7J	10.8J	1.4J	4.1J	7.8J
Dropbox	2.3J	6.3J	21.8J	2.4J	5.5J	14.3J

USB interface which communicates with the serial port module of the power meter through the RS-232C protocol. Appendix C describes this system in detail and introduces the procedure of energy measurement.

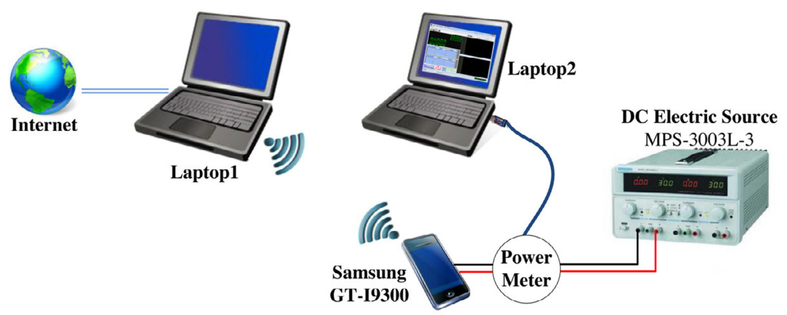
As the energy measurement experiments were conducted in Xiamen University, we could not directly connect to four PCS services located outside China, including Dropbox, Box, GDrive, and OneDrive. Instead, we used a proxy service to access Dropbox and Box for conducting the experiments.

Fig. 7 (b) lists the energy consumption measured from four file operations, including uploading, downloading, deletion and deduplication of files. The result shows that Dbank consumes least energy while Box consumes most energy in most cases. For example, when uploading a renamed 10M file (the original file has been uploaded to the server), Box consumes much energy than others PCS services because Box does not support deduplication. Moreover, Baidu is more energy efficient than Dropbox.

We have also some interesting observations. First, uploading or downloading larger file consumes more energy. But the average energy used for uploading/downloading 1M bytes decreases. Second, file operations on Box and Dropbox usually consume more energy than those on Baidu and Dbank. It may be due to the long RTT between the smartphone and Box/Dropbox. Another possible reason is that Box/Dropbox uses HTTPs while Baidu/Dbank employs HTTP. Thirdly, the energy consumption of file deletion is independent of file sizes because the deletion operation does not involve file transmission.

Compressing files before transmission may save the energy because the amount of traffic is decreased. If a mobile client does not support file compression, its energy consumption during uploading/downloading a *compressible* file will be close to the energy consumption during uploading/downloading an *incompressible* file. We verify it through experiments.

Table 9 records the energy consumption for uploading/downloading compressible files with sizes of 100KB, 1MB, and 10MB. The results shown in Fig. 7(b) come from uploading/downloading incompressible files with the same sizes. We can see that the results in Table 9 are similar to that in Fig. 7(b). For example, Dropbox consumes 21.8J to upload a 10MB compressible



(a) System configuration of energy measurement.

PCS	UPLOAD			DOWNLOAD			DELETE			DEDUP	
	100K	1M	10M	100K	1M	10M	100K	1M	10M	UP(1)	UP(2)
Baidu	2.4J	5.6J	16.4J	1.7J	5.8J	18.1J	1.2J	1.1J	1.1J	16.4J	1.8J
Box	2.8J	8.8J	24.6J	3.3J	9.26J	23.1J	4.0J	3.7J	3.8J	32.8J	24.5J
Dbank	2.1J	4.2J	10.4J	1.1J	3.4J	9.9J	1.6J	1.5J	1.6J	10.4J	1.9J
Dropbox	1.4J	7.1J	21.1J	2.5J	5.38J	14.9J	2.4J	2.3J	2.3J	18.4J	3.2J

(b) Energy consumption in file uploading/downloading/deleting/deduplication.

Fig. 7. Energy measurement system and the measurement results.

file while it consumes 21.1J to upload a 10MB incompressible file. The difference ratio is only 3% (i.e., (21.8J-21.1J)/21.1J). From the energy measurement results, we can also infer that these four mobile clients do not employ file compression. It is in consistent with the result in Table 3, which reveals that these mobile clients do not support file compression through traffic analysis.

4. Discussion

Based on the systematic examinations and comparisons of six PCS services, we have some suggestions to the design of mobile PCS clients. The first eight items concern performance and resource consumption, and the remaining two are potential security issues. These suggestions can also be applied to the design of other similar apps.

1. Deduplication avoids unnecessary transmission, thus saving energy and bandwidth.
2. Although compression may save bandwidth, it may lead to high overhead of CPU/memory and consequently consume more energy. Hence, apps may let customers decide whether such functionality should be enabled or not.
3. Sending multiple files through one TCP connection is more efficient than establishing a new TCP connection for each file, because it takes time for a new TCP connection to reach a large congestion window and fully use the bandwidth. Moreover, the establishment and termination of TCP connections also consume much energy.
4. Chunking helps in error recovery and delta encoding, especially in an unstable mobile network. Without using it, apps have to retransmit the whole file, thus wasting energy and bandwidth. Using HTTP Range option can achieve the same purpose for downloading file. Moreover, the selection of chunk size should consider the network conditions. For example, if the connection is unstable and/or error-prone, a small chunk size is preferred.
5. The period of heartbeat should not be too small for avoiding signaling storm and saving energy/bandwidth.
6. Adjusting TLS/SSL record size to make full use of each TCP packet can lead to sending less packets.
7. Reducing protocol overhead by removing redundant and unnecessary information from headers can save bandwidth and processing time.
8. Employing the CDN technique or deploying multiple servers close to major users can shorten the response time and balance the workload. Moreover, there is a tradeoff between using host name and IP address in realizing load balance. The former makes the system more flexible while the latter avoids performing another round of DNS query/response. However, using IP address instead of host name may allow an attacker to easily launch DDoS attacks targeting on that IP address.
9. HTTPs or other security mechanisms should be used when transmitting sensitive information (e.g., IMEI number, user information).
10. It is preferred to use certificate pinning for defending against MITM attacks.

5. Related work

Although recent research examines several popular PCS services, to our best knowledge [8,9,21,27,39], none of them investigates their mobile applications. Hu et al. conducted the first measurement on Dropbox, Mozy, Carbonite, and CrashPlan in a PC, especially their backup and restore performance [21]. Drago et al. compared five PCS services in terms of client capabilities, protocol design, and data center placement [8]. Li et al. [26] proposed efficient batched synchronization mechanisms for PCS services. Tinedo et al. used the REST APIs provided by three PCS services to measure their transfer speed, variability and failure rate [39].

Some studies focus on specific PCS services. For example, Drago et al. performed an in-depth examination on Dropbox through reverse-engineering its protocols and analyzing data passively collected from different places [9]. Mulazzani et al. revealed security issues in Dropbox, such as unauthorized data access [30]. Han et al. proposed MetaSync file synchronization across multiple untrusted storage services [19]. The application UniDrive [36] used public Web APIs to synergize multiple PCSs to improve communication performance and reliability. Wang et al. observed that using

Table A.10
PCS clients.

PCS	Version of PC client	Version of mobile client
Baidu	3.9.0.2	6.3.0
Box	4.0.4859	2.4.4
Dbank	1.3.2.0	3.1.2.6
Dropbox	2.6.31	2.3.10.4
Gdrive	1.14.6059.0644	1.3.114.26
Onedrive	17.0.4041.0512	2.0.1

Amazon EC2 may limit Dropbox's scalability [40]. Mager et al. examined Wuala from several aspects including infrastructure, data placement, coding techniques, and transport protocol [27].

6. Conclusion

We conduct the first systematic measurement on six popular PCS services from three aspects: protocol, client, and server. By dissecting the protocols and correlating network behaviors at different layers, we not only identify their functionality and features, but also discover new design issues in mobile clients and provide suggest solutions to remedy them. Moreover, we propose and implement a new approach to estimate the response delay of each PCS server. It can facilitate users to select better services and infer the workload of a server.

Acknowledgment

We thank the anonymous reviewers for their quality reviews and suggestions. We also thank Fufeng Chen, Guozhi Xue and Yun Lin (Xiamen University) for their helps in the experiments. This work is supported in part by the Hong Kong GRF/ECS (No. PolyU 5389/13E), the National Natural Science Foundation of China (No. 61202396, 61271242, 61379157). Shenzhen City Special Fund for Strategic Emerging Industries (No. JCYJ20120830153030584), the Scientific Research Fund of Sichuan Provincial Department Science and Technology (No. 2015GZ0333), the Innovation Method Fund of China (No. 2015IM020500).

Appendix A. The versions of PCS clients under investigation

Appendix B. Steps of file uploading and downloading

The file uploading/downloading measures the volume of traffic involved in file uploading/downloading. The steps are as follows and the traffic is captured by TCPDump.

(1) Generate the incompressible and compressible files with different sizes, including 100KB, 1MB, and 10MB.
(2) Upload these files to PCS services using mobile clients. After removing the local files, the clients download these files from the PCS services.
(3) Upload these files to PCS services using PC clients.
(4) Upload these files to PCS services through a browser, and then download files using PC clients.

Step (2) aims at getting the traffic of uploading/downloading files through mobile clients. Step (3) is to obtain the traffic of uploading files through PC clients. Step (4) is conducted for collecting the traffic of downloading files through PC clients.

Appendix C. Details about energy measurement system

We have set up a test bed with a power meter to measure the energy consumption of a smartphone when it accesses these PCS services. In particular, the energy consumption of a smartphone during the processes of uploading, downloading and deleting files of different sizes are analyzed. As shown in Fig C.8, the test bed consists of five elements.

Notebook1 accesses to Internet via a public IP address and provides Wi-Fi service as an AP. It captures all traffic in the test bed by running Wireshark. DC electrical source (MPS-3003L 0 30V) provides a stable voltage for smartphones, which avoids the influences of the unstable voltage when the battery continuously discharges. Power meter is a self-regulating machine to measure the voltage and current values of smartphones. The measurement data can be sent to Notebook2 via serial interface communication in real time. Notebook2 installs the customized software to communicate with the power meter. It obtains the voltage and current values of the smartphone from the serial interface, and then analyzes the operation time and the power consumption of smartphone. The experimental smartphone is Samsung GT-I9300, where the client apps of six PCS services are installed. In this test bed, the power of phone is supplied by the DC electrical source with 3.8V instead of the original battery.

The preparation of energy measurement is to adjust two elements. Firstly, the output voltage of DC electrical source is adjusted to be 3.8000V, which can be verified by a high precision voltage meter. Secondly, we connect the power meter to the DC electrical source. This self-regulating machine adjusts its voltage as 3.8000V and the no-load current as 0.0000A. Then, all elements in the test bed are turned on for energy measurement. The procedure of each experiment is as following.

1. Open one client app in the phone and prepare for file uploading or downloading.
2. Observe the voltage of the phone shown in Notebook2. When the voltage is stable for 5 to 10 seconds, the phone executes the operation to exchange files with PCS service. At the same time, Wireshark in Notebook1 is capturing the traffic.

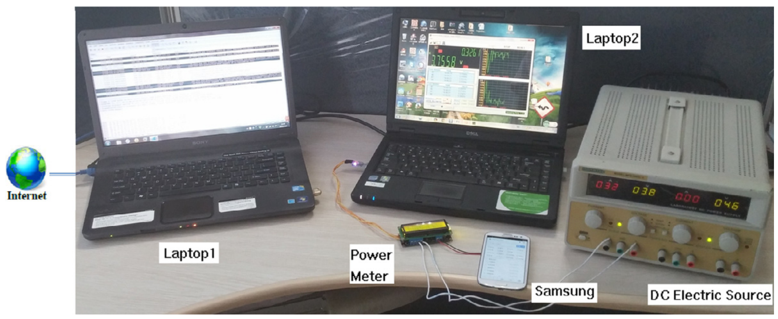


Fig. C.8. The test bed for energy measurement.

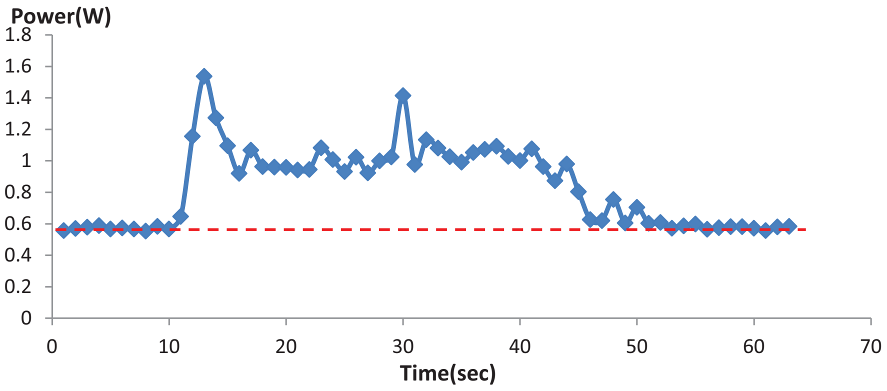


Fig. C.9. Calculation of the energy consumption in Android phone.

3. Analyze the records of traffic captured by Wireshark, and mark the end of file transmission. The voltage is stable after 5 to 10 seconds, and then we turn off the power meter.
4. Calculate the energy consumption of a smartphone according to the power records in Notebook2. The energy consumption of a smartphone caused by the file operation is obtained after eliminating the basic energy consumption, which is caused by the screen and OS. The total power consumption of the phone is shown as Fig. C.9. Here, the basic power consumption is computed as the average of the power values before and after the file operation for a few seconds, shown as the red base line with 0.57261W. Then the energy consumption due to the file operation equals to the area between the power curve and the base line within the operation period.

References

[1] Baidu, Baidu personal cloud storage, 2015, http://goo.gl/DFDkLS.
[2] Baidu, visited at 2015, http://pan.baidu.com.
[3] Box, Box developer api, visited at 2015, https://developers.box.com/.
[4] Box, Inc., visited at 2015, https://www.box.com.
[5] R. Boyd, Getting Started with OAuth2.0, O'Reilly, 2012.
[6] C. Cui, visited at 2015, http://successfultree.blogspot.com/2013/04/tcp-multi-flows-probe-tcpprobe-extension.html.
[7] Dbank, Dbank api, 2015, http://open.dbank.com/.
[8] I. Drago, E. Bocchi, M. Mellia, H. Slatman, A. Pras, Benchmarking personal cloud storage, Proceeding IMC, Barcelona, Spain, 2013.
[9] I. Drago, M. Mellia, M. Munafo, A. Sperotto, R. Sadre, A. Pras, Inside dropbox: understanding personal cloud storage services, Proceeding IMC, Boston, USA, 2012.
[10] Dropbox, Build your app on the dropbox platform, 2015, https://goo.gl/eW0i82.
[11] Dropbox, visited at 2015, https://www.dropbox.com.
[12] S. Fahl, M. Harbach, T. Muders, M. Smith, Why eve and mallory love android: an analysis of ssl (in)security on android, Proceeding ACM CCS, Raleigh, USA, 2012.
[13] R. Fielding, Architectural styles and the design of network-based software architectures, UNIVERSITY OF CALIFORNIA, IRVINE, 2000 Ph.D. thesis.
[14] Google, Google drive api, 2015, https://goo.gl/JQzzxu.
[15] Google, Google drive (gdrive), visited at 2015, https://drive.google.com.
[16] Google, Google play, visited at 2015, https://play.google.com.
[17] Google, Ui automator, visited at 2015, https://developer.android.com/tools/testing-support-library/index.html.
[18] I. Grigorik, High Performance Browser Networking, O'Reilly, 2013.
[19] S. Han, H. Shen, T. Kim, A. Krishnamurthy, T. Anderson, D. Wetherall, Metasync: file synchronization across multiple untrusted storage services, Technical Report, University of Washington, 2014.
[20] HTC Corporation, visited at 2015, http://www.htc.com/us/go/advantage.
[21] W. Hu, T. Yang, J. Matthews, The good, the bad and the ugly of consumer cloud storage, Oper. Syst. Rev. 44 (3) (2010).
[22] J. Huang, F. Qian, Z.M. Mao, S. Sen, O. Spatscheck, Screen-off traffic characterization and optimization in 3g/4g networks, Proceeding IMC, Boston, USA, 2012.
[23] Huawei, Dbank, visited at 2015, http://www.dbank.com.
[24] H. Jiang, C. Dovrolis, Passive estimation of TCP round-trip times, ACM CCR 32 (3) (2002).
[25] Z. Li, C. Jin, T. Xu, C. Wilson, Y. Liu, L. Cheng, Y. Liu, Y. Dai, Z.-L. Zhang, Towards network-level efficiency for cloud storage services, Proceeding IMC, Vancouver, Canada, 2014.
[26] Z. Li, C. Wilson, Z. Jiang, Y. Liu, B.Y. Zhao, C. Jin, Z.-L. Zhang, Y. Dai, Efficient batched synchronization in dropbox-like cloud storage services, Proceeding Middleware, Beijing, China, 2013.
[27] T. Mager, E. Biersack, P. Michiardi, A measurement study of the wuala on-line storage service, Proceeding IEEE P2P, Cambridge, USA, 2012.
[28] MaxMind, Inc., visited at 2015, http://dev.maxmind.com/geoip/legacy/geolite/.
[29] Microsoft, Onedrive, visited at 2015, https://onedrive.live.com.
[30] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, E. Weippl, Dark clouds on the horizon: using cloud storage as attack vector and online slack space, Proceeding USENIX SEC, San Fransisco, USA, 2011.
[31] OneDrive, Onedrive api 2.0, 2015, https://dev.onedrive.com/.
[32] F. Qian, Z. Wang, Y. Gao, J. Huang, A. Gerber, Z. Mao, S. Sen, O. Spatscheck, Periodic transfers in mobile applications: network-wide origin, impact, and optimization, Proceeding WWW, Lyon, France, 2012.
[33] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, O. Spatscheck, Profiling resource usage for mobile applications: a cross-layer approach, Proceeding Mobisys, Washington DC, USA, 2011.
[34] L. Richardson, M. Amundsen, S. Ruby, RESTful Web APIs, O'Reilly, 2013.
[35] SAMSUNG ELECTRONICS CO., LTD., visited at 2015, http://www.samsung.com/global/microsite/galaxys5/mobile/galaxygifts.html.
[36] H. Tang, F. Liu, G. Shen, Y. Jin, C. Guo, Unidrive: synergize multiple consumer cloud storage services, Proceeding Middleware, Vancouver, Canada, 2015.
[37] Team Cymru, Inc., Ip to asn mapping, visited at 2015, https://www.team-cymru.org/Services/ip-to-asn.html.
[38] Telerik, visited at 2015, http://www.telerik.com/fiddler.
[39] R. Tinedo, M. Artigas, A. Moreno-Martinez, C. Cotes, P. Lopez, Actively measuring personal cloud storage, Proceeding IEEE Cloud, Santa Clara Marriott, USA, 2013.
[40] H. Wang, R. Shea, F. Wang, J. Liu, On the impact of virtualization on dropbox-like cloud file storage/synchronization services, Proceeding IWQoS, Coimbra, Portugal, 2012.

Xiapu Luo is a research assistant professor in the Department of Computing and an associate researcher at the Shenzhen Research Institute at the Hong Kong Polytechnic University. His research focuses on mobile networks, smartphone security, network security and privacy, and Internet measurement. Luo has a PhD in computer science from the Hong Kong Polytechnic University. Contact him at csxluo@comp.polyu.edu.hk.

Haocheng Zhou is a student in Department of computer science, Xiamen University, China. His interesting fields are computer networks, mobile applications, network security and software development. Contact him at zhc105@gmail.com.

Yu Le is a research assistant in the Department of Computing at the Hong Kong Polytechnic University. His research focuses on security and privacy with an emphasis on mobile security. Le holds Bachelor's and Master's degrees in information security from Nanjing University of Posts and Telecommunications. Contact him at cslyu@comp.polyu.edu.hk.

Lei Xue is a PhD candidate in the Department of Computing at The Hong Kong Polytechnic University. His research interest includes network security, network measurement and mobile security. Lei received both his B.S. and M.S. in major of information security from Nanjing University of Posts and Communications. You can contact him through cslxue@comp.polyu.edu.hk.

Yi Xie is an Assistant Professor in Xiamen University, China. She received her B.S. and M.S. from Xian Jiaotong University, China, as well as her Ph.D. degree from the Hong Kong Polytechnic University, Hong Kong SAR of China. Her current research interests include high performance communication, network protocol analysis, network security and modeling & simulation. She is a member of ACM and IEEE Communications Society. She is the corresponding author. Contact her at csyxie@xmu.edu.cn.