A Quantitative and Comparative Study of Network-Level Efficiency for Cloud Storage Services

ZHENHUA LI, YONGFENG ZHANG, and YUNHAO LIU, Tsinghua University
TIANYIN XU, University of Illinois at Urbana-Champaign
ENNAN ZHAI, Yale University
YAO LIU, Binghamton University
XIAOBO MA, Xi'an Jiaotong University
ZHENYU LI, Institute of Computing Technology, Chinese Academy of Sciences

Cloud storage services such as Dropbox and OneDrive provide users with a convenient and reliable way to store and share data from anywhere, on any device, and at any time. Their cornerstone is the *data synchronization* (sync) operation, which automatically maps the changes in users' local file systems to the cloud via a series of network communications in a timely manner. Without careful design and implementation, however, the data sync mechanisms could generate overwhelming traffic, causing tremendous financial overhead and performance penalties to both service providers and end users. This article addresses a simple yet critical question: *Is the current data sync traffic of cloud storage services efficiently used?* We first define a novel metric ***TUE*** to quantify the ***T**raffic **U**sage **E**fficiency* of data synchronization. Then, by conducting comprehensive benchmark experiments and reverse engineering the data sync processes of eight widely used cloud storage services, we uncover their manifold practical endeavors for optimizing the TUE, including three intra-file approaches (compression, incremental sync, and interrupted transfer resumption), two cross-file/-user approaches (*i.e.,* deduplication and peer-assisted offloading), two batching approaches (file bundling and sync deferment), and two web-specific approaches (thumbnail views and dynamic content loading). Our measurement results reveal that a considerable portion of the data sync traffic is, in a sense, wasteful and can be effectively avoided or significantly reduced via carefully designed data sync mechanisms. Most importantly, our study not only offers practical, actionable guidance for providers to build more efficient, traffic-economic services, but also helps end users pick appropriate services that best fit their use cases and budgets.

CCS Concepts: • **Information systems** → **Cloud based storage**; Traffic analysis; • **Networks** → *Network performance analysis*;

Additional Key Words and Phrases: Cloud storage service, network-level efficiency, data synchronization, traffic usage efficiency

This work is supported in part by the National Key R&D Program of China under grant no. 2018YFB1004702; the NSFC under grant nos. 61822205, 61471217, 61572475, 61432002, and 61632020; and the Youth Innovation Promotion Association CAS.

Authors' addresses: Z. Li, Y. Zhang, and Y. Liu, School of Software, Tsinghua University, Beijing, China; emails: {lizhenhua1983, zhangyongfeng13, yunhaoliu}@gmail.com; T. Xu, Department of Computer Science and Engineering, University of California, San Diego, CA, US; email: tixu@cs.ucsd.edu; E. Zhai, Department of Computer Science, Yale University, New Haven, CT, US; email: ennan.zhai@yale.edu; Y. Liu, Department of Computer Science, Binghamton University, NY, US; email: yaoliu@binghamton.edu; X. Ma, MOE KLINNS Lab, Xi'an Jiaotong University, Shaanxi, China; email: xma.cs@xjtu.edu.cn; Z. Li, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China; email: zyli@ict.ac.cn.

2376-3639/2019/01-ART3 $15.00
https://doi.org/10.1145/3274526

ACM Reference format:
Zhenhua Li, Yongfeng Zhang, Yunhao Liu, Tianyin Xu, Ennan Zhai, Yao Liu, Xiaobo Ma, and Zhenyu Li. 2019. A Quantitative and Comparative Study of Network-Level Efficiency for Cloud Storage Services. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 4, 1, Article 3 (January 2019), 32 pages.
https://doi.org/10.1145/3274526

1 INTRODUCTION

Cloud storage services such as Dropbox, OneDrive, Google Drive, and iCloud Drive provide users with a convenient and reliable way to store and share data from anywhere, on any device, and at any time. The users' data (e.g., documents, photos, and videos) stored in cloud storage are automatically synchronized across all designated devices (e.g., PCs, tablets, and smartphones) connected to the cloud in a timely manner. With multiplicity of devices – especially mobile devices – that users possess today, such "anywhere, anytime" features significantly simplify data management and consistency maintenance, providing an ideal tool for data sharing and collaboration.

In the past few years, cloud storage services have reached phenomenal levels of success, with the user base growing rapidly. For example, Dropbox has over 500M users who store or update *1.2 billion* files every day and make ~4,000 file edits every second [53]. Also, Google Drive reports that over 800M users have stored more than 2 trillion files using Google's service [27]. In China, despite a late entry into this market in 2012, Baidu Netdisk obtained 400M users in its first 4 years [12].

The key operation of cloud storage services is data synchronization (sync), which automatically maps the changes in users' local file systems to the cloud via a series of network communications, as demonstrated in Figure 1. In a cloud storage service, the user usually needs to assign a designated local folder (called a "sync folder") in which every file operation is noticed and synchronized to the cloud by the client software developed by the service provider. Synchronizing a file involves a sequence of data sync events, such as transferring the data index, data content, sync notification, sync status/statistics, and sync acknowledgment. Naturally, each data sync event incurs network traffic. In this article, this traffic is referred to as *data sync traffic*.

Without careful design and implementation, however, the data sync mechanisms could generate overwhelming data sync traffic, causing tremendous financial overhead and performance penalties to both service providers and end users. From the providers' perspective, the aggregate sync traffic from all users is enormous (given the huge number of files uploaded and modified each day) [42]. To get a quantitative understanding, we analyze a large-scale, ISP-level Dropbox trace [16]. The analysis reveals the following. (1) The sync traffic contributes to more than 90% of the total service traffic, part of which seems to be avoidable. Note that the total service traffic is equivalent to one-third of the traffic consumed by YouTube [15]. (2) Data synchronization of a file (sometimes a batch of files) generates 2.8MB of inbound (client-to-cloud) traffic and 5.18MB of outbound (cloud-to-client) traffic, on average. If Dropbox stores all the data content in Amazon S3[1], according to the pricing policy of S3 [3], the sync traffic would consume nearly \$0.05/GB × *5.18MB* × *1.2 billion* = \$303,500 every day (note that S3 charges only for outbound traffic [3]). These costs grow even further when we consider that *all* cloud storage service providers must bear similar costs, not just Dropbox.

Data sync traffic can also bring considerable (and unexpected) overhead to end users, despite the fact that basic cloud storage services are generally free. News media have reported about user

[1]Dropbox used to store all data content in S3, but started to migrate data content to its own deployed MagicPocket object storage since 2016 [11].

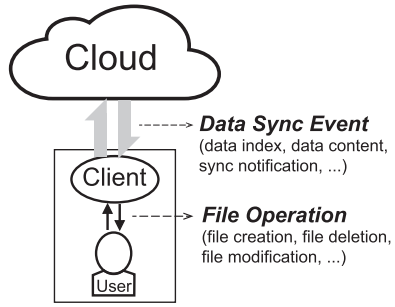

Fig. 1. Data synchronization principle in a cloud storage service.

complaints of unexpected additional charges from ISPs, typically from mobile users with limited data usage caps [28, 50, 57]. As a consequence, some users have warned: "Keep a close eye on your data usage if you have a mobile cloud storage app.'' In addition, some cloud storage applications (e.g., large data backup [55]) are also impaired by the bandwidth constraints between user clients and the cloud. This limitation is regarded as the "dirty secret" of cloud storage services [43]. Hence, users would also benefit from more efficient sync traffic usage.

This article addresses a simple, yet critical, question: *Is data sync traffic of today's cloud storage services efficiently used?* Our goal is to quantify and then optimize the *efficiency* of data sync traffic usage, that is, the pivotal network-level efficiency for cloud storage services, which brings significant benefits for both service provider and end users. For cloud storage service providers, the incentive is to reduce data sync traffic as much as possible to minimize their operational costs (without affecting the customers' experience). For end users of cloud storage services, they desire more efficient traffic use for better quality of experience and lower cost, especially for energy- and traffic-constrained devices such as smartphones.

To answer the question rigorously, we define a novel metric named ***TUE*** to quantify ***T**raffic **U**sage **E**fficiency* of data synchronization. Borrowing a term similar to *PUE* (*Power Usage Effectiveness* $= \frac{\text{Total facility power}}{\text{IT equipment power}}$ [65], a widely adopted metric for evaluating cloud computing energy efficiency), we define

$$TUE = \frac{\text{Total data sync traffic}}{\text{Data update size}}. \tag{1}$$

When a file is updated (created, modified, or deleted) at the user side, the *data update size* denotes the size of altered bits relative to the original local file (or the cloud-stored file if it is not compressed). From the users' point of view, the data update size is an intuitive and natural signifier about how much traffic *should* be consumed. Compared with the absolute value of sync traffic (used in previous studies [5, 14, 15, 26, 30, 31, 36, 68]), TUE better reveals the essential traffic harnessing capability of cloud storage services.

In order to gain in-depth understanding of TUE in practice, we conduct comprehensive benchmark experiments and reverse engineer the data sync processes of eight widely used cloud-storage services: Dropbox, Microsoft OneDrive, Google Drive, iCloud Drive, Box, SugarSync, Seafile, and Baidu Netdisk (Section 3). We examine key impact factors and design choices that are common across all of these services. Impact factors include file size, file operation, data update size, network environment, hardware configuration, access method, and so on. Here 'access method refers to PC client software, web browsers, and mobile apps (via WiFi and 4G). Design choices (of data sync mechanisms) include data compression level, data sync granularity, data deduplication granularity, data delivery protocol (C/S or P2P), interrupted transfer handling (redo or resume),

file bundling and sync deferment (for improved batching), and those web-specific traffic-saving approaches.

Based on deep diving into impact factors and design choices, we extensively unravel the TUE-related characteristics and design trade-offs of these popular cloud storage services. We summarize our measurement results and findings as follows.

We start by studying the intra-file approaches:

- **Compression** [Section 4.2] is a straightforward approach to optimizing TUE. In practice, we find that different levels of compression are applied for different access methods. Specially, the server-side compression level is typically higher than the client side by 5% to 30% (thus, data download usually consumes less traffic than upload for a user client). During the data upload, no service ever compresses data with web-based or mobile apps owing to the concerns of inefficiency of JavaScript and battery consumption of mobile devices in executing computation-intensive operations.
- **Incremental sync** [Section 4.3] can greatly reduce the sync traffic of file edits by transferring only the altered bits or blocks. However, we find that it is supported only by PC clients of a limited number of services and is never available for web-based or mobile apps. The obstacles lie in the RESTful architecture, inefficiency of JavaScript execution inside web-based apps, and energy concerns of mobile apps. On the other hand, our recent research has shown that it can be effectively and efficiently supported with affordable efforts in web-based and mobile apps [63, 64, 67].
- **Interrupted transfer resumption (ITR)** [Section 4.4] can avoid repeated data transfer after a file sync process is interrupted. Our study shows that most services currently support ITR using a variety of block sequences (sequentially or in parallel) and granularities (between 128KB and 32MB). The parallel method can remarkably increase the throughput (by up to 22×), with the cost of more data transfer (2–22 blocks) and traffic waste (up to 100% for a file that is smaller than 96MB) when interruptions happen.

We then study the cross-file/-user approaches:

- **Deduplication (Dedup)** [Section 5.1] can avoid the transmission of data already stored both on the server side and client side. Nevertheless, it is not adopted by the web-based apps of all of the studied services as well as across different users for most services. In particular, we notice that block-level dedup exhibits trivial superiority (in terms of traffic saving) to full-file dedup but incurs much higher computation complexity. Hence, using full-file dedup is generally sufficient to achieve efficient use of sync traffic in realistic scenarios.
- **Peer-assisted offloading (P2P)** [Section 5.2] can considerably cut down the cloud-side traffic cost by around 25% for delivering a popular file shared by multiple users. However, we observe that it is adopted only by Baidu Netdisk, which has a special design for P2P-based cross-user file sharing in which P2P data are transported via UDP (instead of TCP) to enhance throughput and parallelism.

We also study the batching approaches:

- **Bundling** [Section 6.1] small files into a large file can effectively optimize TUE. It has been pervasively used by PC clients and web-based apps but not mobile apps for most services simply because they do not offer an operational user interface via which a user can move a bundle of files into the sync folder all at once.
- **Sync deferment** [Section 6.2]. Frequent edits to files often lead to large TUE; still worse, when handling frequent edits, iCloud Drive and SugarSync degrade from incremental sync

to full-file sync, thus aggravating the traffic overuse. Some services deal with this issue by actively batching the edits using a fixed sync deferment. However, fixed sync deferments are inefficient in certain scenarios. Thus, we propose an adaptive sync deferment (ASD) mechanism to overcome this limitation. In the presence of frequent edits, surprisingly perhaps, we discover that users with relatively "poor" hardware or Internet access save on sync traffic because their edits are naturally batched.

Finally, we study the web-specific approaches:

- **Thumbnail views** (usually in several KBs) [Section 7.1] are used by all services to avoid loading the real content of certain files when the user just needs a brief understanding (or sketches) of the files via web browsers. At present, we observe that all services support thumbnail views for images, over a half support videos, but only OneDrive and Google Drive support documents, as generating thumbnail views for documents (e.g., Microsoft Word) turns out to be more complicated.
- **Dynamic content loading (DCL)** [Section 7.2]. In contrast to thumbnail views that present a simplified sketch, DCL presents an on-demand part of the actual content of a file to the user via the web. We find that it is now supported only by Google Drive, where the pages of a document are dynamically loaded according to user activity. Given that DCL can make the traffic usage proportional to the pages viewed, it should also be adopted by the other services that host numerous documents.

In a nutshell, this article illustrates that for today's cloud storage services, a considerable portion of data sync traffic is, in a sense, wasteful and can be effectively avoided[2] or significantly reduced through careful design and implementation of data sync mechanisms. Here, we unravel the tremendous opportunities and realistic avenues for optimizing network-level efficiency. In particular, our study of TUE provides practical and actionable guidance in two areas: (1) helping service providers build more efficient, traffic-economic cloud storage services; and (2) helping end users select appropriate services that best fit their needs and budgets.

2 BACKGROUND

Section 2.1 describes the basic system architecture of cloud storage services and the typical data sync events. In Section 2.2, we profile their common design framework.

2.1 System Architecture

Mainstream cloud storage services — for example, Dropbox, Baidu Netdisk, and Ubuntu One — typically adopt a two-cloud system architecture [15, 23, 30, 36] as demonstrated in Figure 2. One cloud, the *object storage cloud* (e.g., Amazon S3 and MagicPocket [11]), is employed to host users' file content as well as the small-sized thumbnail views for certain files (if applicable). The other cloud, the *index/control cloud*, is used to maintain users' account information, online status, file metadata (e.g., directories), block index (if applicable), and so forth.

Each instance of the client application exchanges three different types of sync traffic to enable the typical data sync events. First, each client maintains a connection to a *front-end server*. The front-end server authenticates each user's account and stores metadata about the user's files, including the list of the user's files, their sizes and attributes, and pointers to where the files can be

[2]Specifically, duplicate data transfer is mostly avoidable through incremental sync, dedup, and P2P; unnecessary data transfer is mostly avoidable through ITR, sync deferment, and DCL. Avoiding such data transfer usually requires the use of additional control messages and/or metadata whose size is much smaller than that of the avoided data transfer.

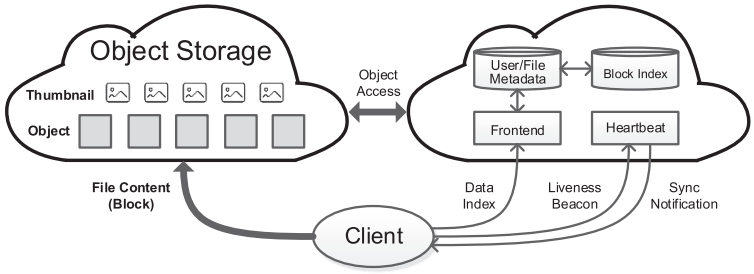

Fig. 2. Architectural overview of a typical cloud storage service system.

found on the object storage cloud. If the user's files are each stored in blocks (in the object storage cloud), there is also a *block index* coupled with the *user/file metadata index*.

Second, each client sets up a connection to an object storage server for transferring the file content or file blocks. If the file is large, the client may need to set up more connections to multiple object storage servers for parallel transfer (thus significantly increasing the transfer throughput). Moreover, if data compression is supported by the client, the client will compress files or file blocks before uploading them to the object storage cloud. In addition, if there are modifications to synced files and incremental sync is supported by the client, the client will first resort to a local cache or contact the front-end server for calculating the differences between the synced files and updated files, and then upload such differences to the object storage cloud as (compressed) binary diffs.

Third, each client maintains a connection to a *heartbeat server*. Periodically, the client sends a liveness beacon to the heartbeat server to report its online status and receives sync notifications from the index/control cloud (e.g., a shared file is modified by another user and should be re-synced).

In addition to these three types of sync traffic, there is other traffic exchanged between the two clouds to enable the sync processes, referred to as the *inter-cloud traffic*. For example, when incremental sync is executed by the client and the front-end server, the front-end server may have to read corresponding data from the object storage cloud for calculating the file differences. Further, for some cloud storage services, the front-end server may need to re-compress users' files with a higher compression level (than the client side). In this case, the front-end server also needs to write novel data into the object storage cloud. However, since the inter-cloud traffic is completely invisible to users and often goes through dedicated network connections (e.g., LAN connections inside the same data center [61]), it is not counted as the data sync traffic in our research.

2.2 Common Design Framework

From the perspective of TUE, the common design framework of cloud storage services involves a number of impact factors and design choices on the client side, server (cloud) side, or network side. Impact factors include objective factors such as client location, hardware, file size, data update size, network environment, and the like that must be accounted for in the design and use of cloud storage services. Design choices include subjective design decisions that the system designers make, such as data sync granularity, data compression level, data deduplication granularity, and so forth. To avoid being trapped by trivial or elusive issues, we select key impact factors and design choices according to the following two rules:

- *Rule 1: The impact factors should be relatively constant or stable so that our research results can be easily repeated.*

Table 1. Key Impact Factors and *Design Choices*

Client side	Client location Client hardware Access method File size File operation Data update size Data update rate *Data compression level* *Sync deferment*
Server side	*Data sync granularity* *Data dedup granularity* (*Data compression level*)* *Thumbnail view support* Dynamic content loading
Network side	Sync traffic Bandwidth Latency Wired/WiFi/4G connection *Data delivery protocol*

*Note: The server-side data compression level may be different from the client-side.

- *Rule 2: The design choices should be measurable and service/implementation independent to make our research methodology widely applicable.*

Following Rule 1, we do not study impact factors such as the sync delay (which measures how long the client synchronizes a file to the cloud) and cloud server location. For example, we observe that uploading a 1MB JPEG photo to Google Drive may incur an elusive sync delay varying between several seconds and minutes (under different network environments). Instead, we choose to study the *sync traffic*, which is almost invariable in all cases.

Following Rule 2, we do not consider design choices such as metadata structures, file segmentation, and replication on the cloud side because they require specific or even confidential knowledge of the back-end cloud implementation.

In the end, we select to study the key impact factors and design choices as listed in Table 1. Some are self-explanatory or have been explained before. Below, we further explain a few:

- *Data update rate* denotes how often a file operation happens.
- *Sync deferment*. When frequent file edits occur, some cloud storage services intentionally defer the sync process for a certain period of time for batching the file updates.
- *Data sync granularity*. A file operation is synchronized to the cloud either in a *full-file*, *block-level*, or *chunk-level* granularity. When the full-file sync is adopted, the whole updated file is delivered to the cloud; otherwise, only those file blocks/chunks that contain altered bits (relative to the file stored in the cloud) or simply the altered bits are delivered.
- *Data dedup granularity* denotes the *unit* (a full file or a file block) at which data fingerprints are computed and compared to avoid delivering duplicate data units to the cloud. Note that data dedup can be performed across different files owned by different users.
- *Chunk size* and *block size* can be ambiguous in a good amount of the literature. In our article, however, *chunk size* and *block size* are distinct in terms of both application scenarios and typical sizes. *Chunk size* is mentioned only for intra-file delta sync (Section 4.3), and it is typically just several KBs. In contrast, *block size* is mentioned for either intra-file CDC-based sync (Section 4.3) or inter-file deduplication, and it is typically tens to hundreds of KBs or several MBs.
- *Bandwidth* is defined as the peak upload rate between the client and the cloud server. We measure it by uploading a large file to the cloud and recording the network traffic with the Wireshark network protocol analyzer. Although bandwidth is not constant on the Internet (due to the intrinsic dynamics of the Internet), we are able to control it in a fine-grained manner by using Netfilter/Iptables (which will be discussed in detail in Section 3.1).
- *Latency* is defined as the round trip time (*RTT*) between the client and the cloud. We measure it by using the standard `Ping` command. Similarly, although latency is not constant on the Internet, we are able to control it in a fine-grained manner by using Netfilter/Iptables.

3 METHODOLOGY

In this section, we first design a variety of benchmark experiments to study the TUE of cloud storage services (Section 3.1) and then reverse engineer their data sync processes to uncover their practical endeavors for optimizing the TUE (Section 3.2).

3.1 Benchmark Experiments

Our designed benchmarks span multiple commercial cloud storage services and make use of three datasets involving diverse client devices at distinct locations and network environments.

Cloud Storage Services. Among today's dozens of commercial cloud storage services, our research focuses on the following eight services: Dropbox, OneDrive, Google Drive, iCloud Drive, Box, SugarSync, Seafile, and Baidu Netdisk, as they are either the most popular (in terms of user base) or the most representative (in terms of data sync mechanism).

Client Locations and Network Connections. Since the vast majority of the above cloud storage services are deployed in the United States, we select two distinct locations to perform each experiment: SF (San Francisco, United States) and BJ (Beijing, China). In SF, the client device is connected to the Internet with 20Mbps bandwith; in BJ, the Internet bandwidth is restricted to within 1.6Mbps. Thus, in a *coarse-grained* manner, SF represents a location close to the cloud with fine network connections, while BJ represents a location remote from the cloud with poor network connections.

Controlled Bandwidth and Latency. To tune the network environment in a *fine-grained* manner, we interpose a pair of packet filters (using Netfilter/Iptables) between the client and the cloud. These filters enable fine-grained adjustment of bandwidth and latency in either direction, that is, restricting them within or close to given thresholds: for example, bandwidth $\leq$ 10Mbps and latency $\geq$ 100msec.

Datasets. Three datasets are used in our research to guide the design of benchmark experiments, all of which are publicly available. The first dataset (`Dropbox-dataset`) is a large-scale Dropbox dataflow trace collected at the ISP level [15] and is available at https://www.simpleweb.org/wiki/index.php/Traces#Cloud_Storage. The second dataset (`PC-dataset`) records detailed information of 222K files in 153 PC users' local sync folders [33] and is available at http://www.greenorbs.org/people/lzh/public/traces.zip. The involved services are Google Drive, OneDrive, Dropbox, Box, UbuntuOne, and SugarSync. The third dataset (`Mobile-dataset`) consists of HTTP-level request logs from the front-end servers of a commercial mobile cloud storage service very similar to Google Drive [35] and is available at http://fi.ict.ac.cn/data/cloud.html.

Controlled File Operations. Guided by the characteristics of the three abovementioned datasets, we synthetically generate various file operations, including file creations, deletions, edits, and frequent edits. These operations are applied upon both compressed and compressible files.

Client Devices and Software. Ten client devices are employed in the experiments: 5 in SF (S1–S5) and 5 in BJ (B1–B5). Their detailed hardware information is listed in Table 2. S1/B1 represents a typical PC at the moment, S2/B2 an outdated PC, S3/B3 an advanced PC with SSD storage, S4/B4 a typical Android smartphone, and S5/B5 is an iPhone 7. S1–S3 and B1–B3 are installed with Windows 10 and the Chrome-58.0 web browser. For each concerned cloud storage service, we use its latest client software as of September 2017 – Dropbox 44.4, OneDrive 18.025, Google Drive 3.39, iCloud Drive for macOS 10.13, Box 4.0, SugarSync 3.9, Seafile 6.1, and Baidu Netdisk 5.7. Additionally, we archive these client versions at https://www.dropbox.com/s/no21kflue1wrihi/clients.zip?dl=0 to facilitate other researchers easily repeating our experiments and reproducing our results.

Table 2. Hardware Information of the Experimental Client Devices

Machine	CPU	Memory	Disk Storage
PC: S1 @ SF	4-core Intel i5 @ 1.70GHz	4GB	7200RPM
PC: S2 @ SF	Intel Atom @ 1.00GHz	1GB	5400RPM
PC: S3 @ SF	4-core Intel i7 @ 3.10GHz	8GB	SSD
Phone: S4 @ SF	4-core ARM @ 2.50GHz	4GB	Flash
Phone: S5 @ SF	4-core A10 @ 2.34GHz	2GB	Flash
PC: B1 @ BJ	4-core Intel i5 @ 1.70GHz	4GB	7200RPM
PC: B2 @ BJ	Intel Atom @ 1.00GHz	1GB	5400RPM
PC: B3 @ BJ	4-core Intel i7 @ 3.10GHz	8GB	SSD
Phone: B4 @ BJ	4-core ARM @ 2.50GHz	4GB	Flash
Phone: B5 @ BJ	4-core A10 @ 2.34GHz	2GB	Flash

3.2 Reverse Engineering the Data Sync Process

Seeking for a comprehensive and in-depth understanding of the TUE optimization mechanisms used by the studied services, we decompose their data sync processes through sync traffic recording, IP address tracking, HTTP parsing, HTTPS cracking, and source-code reviewing.

First, all of the sync traffic (incoming/outgoing IP packets) generated in the benchmark experiments are recorded using Wireshark. For the Android/iOS smartphones, we route the traffic through a PC that promiscuously monitors the IP packets using Wireshark. With this network-level information, we can directly get or calculate the client/server IP addresses, traffic size, and duration of a concerned data sync process.

Second, we track the IP addresses of involved servers to figure out their geo-locations, domains, owners, and the like based on the IP lookup service *IP-adress.com*. With such information, we can unravel the coarse-grained system architecture of a cloud storage service. For instance, we get the knowledge that Dropbox is using two different clouds to store the file content and metadata; moreover, Dropbox has deployed multiple data centers across the world for location-aware data transfer.

Third, we analyze the HTTP content carried in the sync traffic to dig out more useful information. Usually, we can determine the HTTP protocol version, connection type (e.g., Keep-Alive), host name of the connected server, application data size, and so forth.

Fourth, because all of the studied services have encrypted most application data using HTTPS (more specifically, TLS), we attempt to crack TLS communication by launching man-in-the-middle attacks using the Charles web debugging proxy. It turns out that we can crack the HTTPS content of OneDrive, Box, and Seafile. For the three services, we are able to get detailed information of each synced file, such as its ID, creation time, modification time, concrete content (i.e., the most critical and confidential detail), and the IP address of its residential server. For the remaining five services, we are unable to crack since their clients do not accept the root CA certificates forged by Charles. Note that for all services, TLS-related traffic overhead is included in our calculation of sync traffic.

Finally, we thoroughly understand the working process of Seafile by reviewing its public source code. Thereby, we acquire more detailed information, in particular, its use of content-defined chunking (CDC) [48], which is more dynamic and complicated than traditional fixed-size chunking. This greatly facilitates our analysis of Seafile's measurement results in Sections 4.3, 4.4, 5.1, and 6.2.

3.3 Overview of Our Major Findings

Based on the above methodology, we are able to thoroughly unravel the TUE relevant characteristics, design trade-offs, and optimization opportunities of the 8 mainstream cloud storage services. Detailed research results will be presented in the remainder of the article. As an overview and a roadmap of our research results, Table 3 summarizes the major findings and their implications.

4 INTRA-FILE APPROACHES

This section presents our measurement results on TUE when a simple operation is performed on a single file. Based on the results, we discover three intra-file approaches to optimizing TUE.

4.1 File Creation and Deletion

Experiment 1: We first study the simplest case of creating a highly compressed file of Z bytes inside the sync folder. Thereby, calculating the TUE of file creation becomes straightforward (TUE $= \frac{\text{Total sync traffic}}{Z \text{ bytes}}$). According to the `PC-dataset` described in Section 3.1, most compressed files are small (average = 732KB, median = 3.2KB), and the maximum compressed file size (1.97GB) is below 2.0GB. Hence, we experiment with $Z \in \{1, 1K, 10K, 100K, 1M, 10M, 100M, 1G\}$.

The second goal of Experiment 1 is to get a quantitative understanding of the *overhead traffic*, as TUE heavily depends on the ratio of the overhead traffic over the total sync traffic. Synchronizing a file to the cloud always involves a certain amount of overhead traffic, which arises from TCP/HTTP(S) connection setup and maintenance, metadata delivery, and the like. Specifically, the overhead traffic is equal to the total sync traffic excluding the payload traffic for delivering the file content; thus, in Experiment 1, Overhead traffic ≈ Total sync traffic - Z bytes.

Table 4 lists the results of Experiment 1. We vary the file size from 1B to 1GB, but for brevity list only four typical sizes: 1B, 1KB, 1MB, and 10MB. The table records the sync traffic generated by the three typical service access methods: PC client, web (browser) based, and mobile app (via WiFi and 4G). In general, from Table 4 we have the following finding: TUE for synchronizing a (compressed) file creation mainly depends on the file size—a *small* file results in big TUE up to 84,000, while a *big* file incurs small TUE approaching 1.0.

This finding poses a key question: *What is a small size and what is a moderate size?* By plotting the TUE versus File Size relationship (for PC clients) in Figure 3, we get an intuitive conclusion that a moderate size should be at least 100KB and must exceed 1MB in order to achieve small TUE—at most 1.5 and must stay below 1.2. Here, we draw the curve only for PC clients since the corresponding curves for web-based and mobile apps are similar.

Experiment 2: Each file created in Experiment 1 is deleted after it is completely synchronized to the cloud to acquire the sync traffic information of a file deletion.

The results of Experiment 2 indicate that deletion of a file usually generates negligible (< 100KB) sync traffic regardless of the cloud storage service, file size, or access method. The reason is straightforward: when a file f is deleted in the user's local sync folder, the user client notifies the cloud to change only some attributes of f rather than remove the content of f. In fact, such "fake deletion" also facilitates users' data recovery, such as the version rollback of a file.

4.2 Data Compression

Experiment 3: To study whether data updates are compressed before they are synchronized to the cloud, we create an X-byte text file inside the sync folder. As a small file is hard to compress, we experiment with X = 1M, 10M, 100M, and 1G. Each text file is filled with random English words. If data compression is actually used, the resulting sync traffic should be obviously less than the

Table 3. Our Major Findings, Their Implications, and Locations of Relevant Sections

Intra-File Approaches	Implications
Section 4.1 (File creation): Creating a small file (<100KB) results in big TUE up to 84,000, while a big file (>1MB) incurs small TUE approaching 1.0.	(File deletion): Deleting a file incurs negligible sync traffic; thus, a user does not need to worry about the traffic for a file deletion.
Section 4.2 (Compression): Server-side compression level is typically higher than client-side by 5%–30%. During the data upload, no service ever compresses data with web-based or mobile apps.	Uploading a file usually consumes more traffic than downloading it. When uploading a file, the user had better use a PC client rather than a web browser or a mobile app.
Section 4.3 (File Edit): Incremental sync is supported only by PC clients of several services and never available for web-based or mobile apps owing to inefficiency of JavaScript execution inside web-based apps or energy concerns of mobile apps.	Most of today's cloud storage services are built on top of RESTful infrastructure that support data access only at *full-file* level. TUE can be significantly improved by implementing incremental sync with an extra mid-layer.
Section 4.4 (Interrupted transfer resumption): Most services currently support ITR using a variety of block sequences (sequentially or in parallel) and granularities (between 128KB and 32MB).	The parallel method can remarkably increase the throughput (by up to 22×), with the cost of more data transfer (2–22 blocks) and traffic waste when interruptions occur.
Cross-File/User Approaches	**Implications**
Section 5.1 (Deduplication): Dedup is not adopted by web-based apps of all of the studied services as well as across different users for most services.	Using full-file dedup is generally sufficient to achieve efficient use of sync traffic in realistic scenarios.
Section 5.2 (Peer-assisted offloading): P2P can considerably cut down the cloud-side traffic cost by ~25% but is only adopted by Baidu Netdisk with a special design to enhance throughput and parallelism.	As cloud storage-based content sharing is becoming increasingly popular, we suggest that the other services also adopt P2P to achieve cost–benefit trade-offs.
Batching Approaches	**Implications**
Section 6.1 (Bundling): Bundling small files into a large file can effectively optimize the TUE but has not been used by the mobile apps of most services.	An operational UI should be offered in mobile apps via which a user can move a bundle of files into the sync folder all at once.
Section 6.2 (Sync deferment): Frequent edits to a file often lead to large TUE. Some services deal with this issue by actively batching the edits using a fixed sync deferment.	For providers, we demonstrate that an *adaptive sync deferment* (ASD) mechanism that dynamically adjusts the sync deferment is superior to a fixed sync deferment.
Section 6.3 (Network and hardware): Users with poor hardware or Internet access save on sync traffic, as their frequent edits are naturally batched.	In the presence of frequent edits, today's cloud storage services actually bring good news (in terms of TUE) to such users.

(Continued)

Table 3. Continued

Web-Specific Approaches	Implications
Section 7.1 (Thumbnail views): All services support thumbnail views for images, over half support videos, but only two support documents.	Generating thumbnail views for documents (e.g., Microsoft Word) seem more complicated; thus, it can be performed offline on servers.
Section 7.2 (Dynamic content loading): DCL is now supported only by Google Drive, where the pages of a document are dynamically loaded.	Given that DCL can make traffic use proportional to the pages viewed, it should also be adopted by the other services.

Table 4. Sync Traffic of a Compressed File Creation

Service	PC client sync traffic (Bytes)				Web-based sync traffic (Bytes)			
	1	1K	1M	10M	1	1K	1M	10M
Dropbox	9K	7K	1.11M	11.0M	18K	11K	1.06M	10.6M
OneDrive	6K	7K	1.06M	10.6M	17K	17K	1.10M	10.8M
Google Drive	4K	4K	1.15M	12.1M	30K	12K	1.49M	10.9M
iCloud Drive	15K	23K	1.08M	10.7M	41K	38K	1.08M	10.3M
Box	2K	5K	1.13M	11.2M	49K	37K	1.12M	11.8M
SugarSync	23K	21K	1.12M	12.2M	38K	13K	1.09M	11.0M
Seafile	4K	5K	1.05M	10.9M	3K	3K	1.05M	10.4M
Baidu Netdisk	24K	26K	1.11M	11.5M	34K	36K	1.15M	11.8M

Service	Mobile app sync traffic (Bytes), WiFi / 4G			
	1	1K	1M	10M
Dropbox	10K / 23K	10K /24 K	1.12M / 1.09M	11.0M / 10.9M
OneDrive	8K / 30K	9K / 22K	1.06M / 1.10M	10.6M / 11.0M
Google Drive	6K / 36K	6K / 24K	1.37M / 1.11M	12.9M / 10.6M
iCloud Drive	54K / 84K	47K / 87K	1.07M / 1.10M	10.1M / 10.1M
Box	3K / 11K	4K / 13K	1.12M / 1.10M	11.3M / 11.0M
SugarSync	12K / 34K	12K / 35K	1.24M / 1.10M	11.5M / 10.7M
Seafile	5K / 8K	6K / 9K	1.04M / 1.07M	10.8M / 10.8M
Baidu Netdisk	43K / 46K	54K / 51K	1.13M / 1.15M	10.9M / 10.6M

original file size. Furthermore, after each text file is completely synchronized, we download it from the cloud with a PC client, a web browser, and a mobile app, respectively, to examine whether the cloud delivers data updates in a compressed form.

As a typical case, the results corresponding to a 10MB text file are listed in Table 5. In the file upload (UP) phase, only Dropbox, Google Drive, iCloud Drive, and Seafile compress data with PC clients. No service ever compresses data with web browsers or mobile apps. The motivation of such a difference is intuitive: web browsers severely suffer from the inefficiency of JavaScript in executing computation-intensive operations (e.g., data compressions) [63], and mobile devices might suffer from the battery consumption of data compression. This shortcoming might be partially overcome with existing techniques such as Migratory Compression [38], which can help common compressors (e.g.,. `gzip`) better and faster compress a file or a batch of files via coarse-grained data reordering.

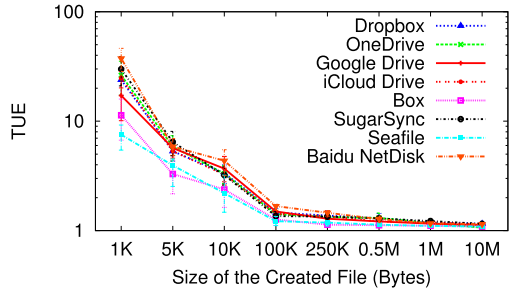

Fig. 3. Relationship between the TUE in log scale and the size of the created file.

Table 5. Sync Traffic of a 10MB Text File Creation

Service	Sync traffic (MB)					
	PC client		Web-based		Mobile app, WiFi / 4G	
	UP	DN	UP	DN	UP	DN
Dropbox	**5.0**	**4.8**	10.4	**8.1**	10.8 / 11.4	**8.4** / **7.9**
OneDrive	10.3	10.8	10.3	**8.0**	11.3 / 11.0	11.3 / 12.9
Google Drive	**7.4**	10.7	10.7	10.8	10.8 / 11.2	11.0 / 10.9
iCloud Drive	**4.6**	**4.4**	10.9	11.2	10.1 / 10.3	10.4 / 10.5
Box	10.4	10.5	10.3	**8.0**	10.8 / 11.2	**8.1** / **7.7**
SugarSync	10.7	10.5	10.2	**8.0**	10.8 / 11.0	10.6 / 10.2
Seafile	**5.5**	10.5	10.4	10.5	11.3 / 10.9	10.5 / 10.2
Baidu Netdisk	10.5	11.3	10.8	10.5	10.6 / 10.4	11.0 / 10.8

UP: The user uploads the file to the cloud. DN: The user downloads the file from the cloud. The bold type denotes the cases when the sync traffic is smaller than the file size, which thus implies the existence of data compression.

Further, we observe that the 10MB text file can be compressed to nearly 4.4MB using the highest-level WinZip compression on our desktop. Thus, we conclude that the iCloud Drive has made its best efforts in data compression, while the compression level of Google Drive is quite low.

Next, in the download (DN) phase, Dropbox compresses data with all access methods; Box compresses data with web browsers and mobile apps; while OneDrive, iCloud Drive, and SugarSync compress data with only a single access method. Most importantly, in the above cases, the compression level is generally higher than that in the upload (UP) phase by around 5% to 30%, because the former is adopted by relatively powerful cloud servers while the latter is adopted by client devices. Consequently, for a compressible file, downloading it from the cloud usually consumes less traffic than uploading it to the cloud.

4.3 File Edit and Incremental Sync

Experiment 4: File edits are frequently made by cloud storage users [33, 36]. This section studies a simple case of file edits: editing a random byte in a compressed file of Z bytes inside the sync folder (we are directly modifying a compressed file here rather than compressing a file after editing). In this case, TUE $= \frac{\text{Total sync traffic}}{\text{1Byte}}$. Similar to what we discussed in Section 4.1, we experiment with $Z \in \{1, 1\text{K}, 10\text{K}, \ldots, 1\text{G}\}$ and plot the sync traffic of four typical sizes: 1K, 10K, 100K, and 1M in Figure 4.

Figure 4 shows that today's cloud storage services generally use three kinds of data sync granularities: (1) full-file, (2) block-level, and (3) chunk-level. Accordingly, their data sync mechanisms

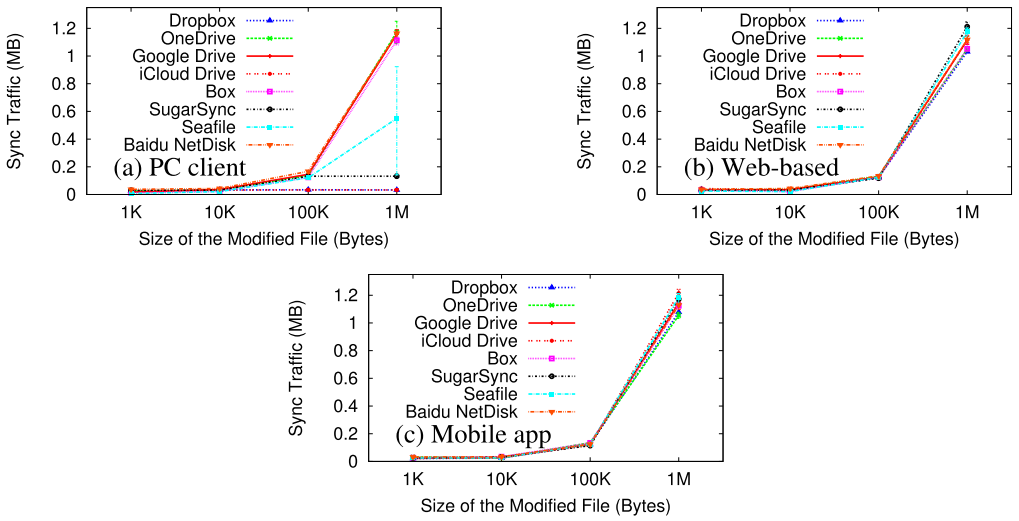

Fig. 4. Sync traffic of a random byte edit, corresponding to the three typical access methods.

are classified into *full-file sync*, *CDC-based sync*, and *delta sync*. The latter two (CDC-based sync and delta sync) are collectively referred to as *incremental sync*.

Google Drive is an example of the use of *full-file sync*. When a random byte is edited in a Z-byte compressed file, the resulting sync traffic is almost the same as that of creating a new Z-byte compressed file. In other words, Google Drive deals with each file edit by simply uploading the full content of the modified file to the cloud and then deleting the old file. Hence, Google Drive is more suitable for hosting media files (such as photos, music, and videos), which are rarely edited by users.

Seafile (PC client) is an example of the use of *CDC-based sync*. When the file size is small ($\leq$ 100KB) it looks like full-file sync; when the file size reaches 1MB, the sync traffic is nearly 0.55MB, on average, varying between 0.14MB and 0.9MB. To demystify the issue, we review the public source code of Seafile [41] and find that this is attributed to Seafile's use of CDC [48], which is more dynamic and complicated than traditional fixed-size chunking — the sync traffic depends on the specific chunking scheme of a file and the chunking scheme depends on the specific content of the file [6, 14, 47].

Dropbox (PC client) is an example of the use of *delta sync*. When a random byte edit occurs, the resulting sync traffic stays around 50KB, regardless of the size of the modified file. According to the working principle of the de facto delta sync protocol rsync [56], when a user edits a file from f to f', the client sends a request to the server to kick out the process of delta sync. On receiving the request, the server first executes fixed-size chunk *segmentation* and *fingerprinting* operations on f and then returns a *checksum list* of f to the client. After that, based on the checksum list of f, the client first performs chunk *search* and *comparison* operations on f' and then generates both the *matching tokens* and *literal bytes*. The matching tokens indicate the overlap between f and f', while the literal bytes represent the novel parts in f' relative to f. Consequently, once a random byte is changed in a file f, in most cases, the whole data chunk that contains this byte must be delivered for synchronizing f. Therefore, the sync granularity (i.e., the chunk size C) can be approximately estimated as $C \approx$ Total sync traffic − Overhead traffic. From the results of Experiment 1, we understand that the overhead traffic of synchronizing a 1B file with the Dropbox PC client is nearly 40KB. Therefore, the data sync granularity of Dropbox PC client is estimated as:

$C \approx 50\text{KB} - 40\text{KB} = 10\text{KB}$. This is further validated by the recommended default chunk size (*i.e.,* from 700B to 16KB) in the original rsync implementation [7]. Moreover, we find that iCloud Drive and SugarSync also use delta sync for their PC clients, and SugarSync seems to use a larger chunk size at around 100KB.

Although enabling incremental sync can significantly help reduce the sync traffic of file edits, implementing it is not an easy job in practice. Most of today's cloud storage services (e.g., Dropbox) are built on top of RESTful infrastructure [18] (e.g., Amazon S3 and MagicPocket [11]). To simplify both providers' implementation complexity and developers' programming complexity, RESTful infrastructure [17] typically supports data access operations only at the *full-file* level, such as PUT (upload a new file), GET (download a whole file), and DELETE (delete a whole file). Thus, enabling incremental sync usually requires an extra mid-layer for transforming MODIFY into GET + PUT + DELETE in an efficient manner (such as what Dropbox has done [15, 36]). Since file edits frequently occur, implementing such a mid-layer is worthwhile for improved network-level efficiency.

In contrast, as in Figure 4(b) and Figure 4(c), web-based apps and mobile apps for all services are still using full-file sync, owing to the inefficiency of JavaScript execution inside web-based apps [63] and the energy concerns with mobile apps. However, as demonstrated in recent research [13, 63, 64, 67], these obstacles can be practically addressed affordably. In other words, there is still plenty of space for optimizing network-level efficiency using incremental sync.

4.4 Interrupted Transfer Resumption (ITR)

Experiment 5: The dynamic nature of the Internet implies that network disconnections can happen to cloud storage clients occasionally. ITR is known to be an effective approach to mitigating the negative effect caused by network disconnections, especially during the transfer of a large file. To clarify whether ITR is employed in cloud storage services, we first assign the client to sync a 100MB file to the cloud and then turn off the network connection when the first half (50MB) of the file has been synced. Ten minutes later, we resume the network connection to see what happens to the file sync process.

Our experiment results show that most services currently support ITR using a block-based approach; in particular, OneDrive and Baidu Netdisk support ITR for all access methods. Specifically, when the client wants to sync a large file to the cloud, it first divides the file into a few blocks [14, 15, 23, 35] and then syncs these blocks to the cloud respectively in a sequential or parallel manner. In this way, network disconnections impair the transfer of only the block(s) being synced, thus effectively restricting the negative influence brought by network disconnections.

As listed in Table 6, ITR is implemented with different block sequences (sequentially or in parallel) and granularities (ranging from 128KB to 32MB). OneDrive's PC clients employ the Windows BITS Upload protocol [46] to sequentially sync a large file in 16MB blocks. In comparison, Seafile's PC clients simultaneously sync multiple blocks (in different sizes) of a large file to the cloud. The parallel method can remarkably increase the throughput (by up to 22×), with the cost of more data transfer (2–22 blocks) and traffic waste (up to 100% for a file that is smaller than 96MB) when interruptions occur.

We also find in Table 6 a notable phenomenon that parallel transferring of data blocks is supported only by PC clients and not by web browsers or mobile apps. The technical rationale behind this phenomenon is to avoid possible stagnation or even hanging of web browsers caused by the high computation burden of parallel transferring. For mobile cloud storage apps, there are no essential technical obstacles that prevent parallel transferring (given that many mobile apps of other applications can actually support parallel transferring) but for computation and energy concerns.

Table 6. Block Sequences and Granularities for Different ITR Implementations in Our Studied Services

Service	Block sequence			Block granularity		
	PC client	Web-based	Mobile app	PC client	Web-based	Mobile app
Dropbox	Parallel (≤ 4)	–	Sequential	4MB	–	4MB
OneDrive	Sequential	Sequential	Sequential	16MB	128KB - 16MB	8MB
Google Drive	–	–	Sequential	–	–	$<$ 256 KB
iCloud Drive	Parallel (≤ 3)	–	–	1MB - 32MB	–	–
Box	–	–	–	–	–	–
SugarSync	Sequential	–	–	6MB	–	–
Seafile	Parallel (≤ 3)	–	–	128KB - 4MB	–	–
Baidu Netdisk	Parallel (≤ 22)	Sequential	Sequential	4MB	4MB	4MB

5 CROSS-FILE/USER APPROACHES

There is considerable data duplication across different files and users, which can be exploited to optimize the TUE. Also, data reuse among end users can minimize the cloud-side traffic cost for delivering a popular file. This section investigates what approaches have been taken by today's cloud storage services to achieve cross-file/user data sync optimizations.

5.1 Data Deduplication

Experiment 6: Deduplication can avoid the transmission of data already stored both on the server side and client side. Inferring the deduplication granularity of a cloud storage service requires some effort, especially when the deduplication block size B (bytes) is not a power of two (i.e., $B \neq 2^n$, where n is a positive integer). To measure the deduplication granularity, we design and implement Algorithm 1 (named the Iterative Self Duplication Algorithm). It infers the deduplication granularity by iteratively duplicating and uploading one or multiple synthetic file(s) and analyzing the incurred data sync traffic[3]. It is easy to prove that the iteration procedure can finish in $O(\log(B))$ rounds. Nevertheless, given that each round may last for a long period of time, we execute Algorithm 1 for all eight concerned cloud storage services under normal network environments and plot their real-world time consumptions in Figure 5. Even in the worst case (iCloud Drive), Algorithm 1 takes only 210s, thus demonstrating its practicality.

First, we study inter-file data deduplication with respect to an identical user account. By applying **Experiment 6** to the concerned cloud storage services, we determine their data deduplication granularity in Table 7 (2nd column). In this table, "Full file" means that data deduplication occurs only at the full-file level, "4MB" (for Dropbox) indicates that the deduplication block size $B = 4$MB, and "–" shows that there is no deduplication performed. Note that block-level deduplication naturally implies full-file deduplication but *not* vice versa.

Second, we study cross-user data deduplication. For each cloud storage service, we first upload a file f to the cloud and then use another user account to upload f to the cloud again. In this case, the sync traffic should be trivial if full-file deduplication is performed across users. If the cross-user full-file deduplication is confirmed, Experiment 6 is run again todetermine the accurate cross-user deduplication granularity; otherwise, we can conclude that there is no cross-user data deduplica-

[3]To deal with possible collision of content in different files generated (though the practical collision might be extremely low), we leverage `rsync` (the de facto delta sync protocol) to check for common content among the different files generated in a fine-grained manner (i.e., at a KB level). In all checks, we did not find any collision of content.

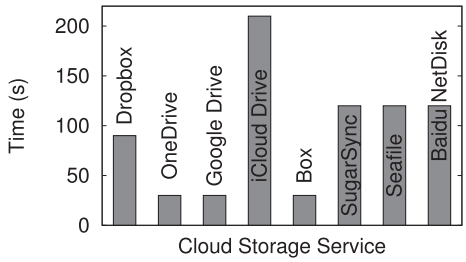

Fig. 5. Real-world execution time of Algorithm 1 for our studied cloud storage services.

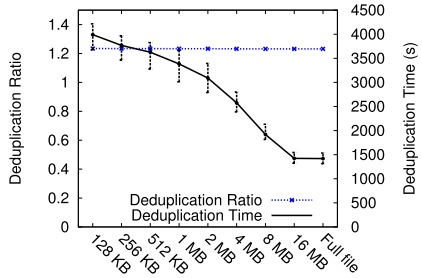

Fig. 6. Deduplication ratio and time as the block size varies between 128KB, . . . , 16MB, and full file.

Table 7. Data Deduplication Granularity

Service	Same user	Cross users
	PC client & Mobile app	PC client & Mobile app
Dropbox	4MB	–
OneDrive	–	–
Google Drive	–	–
iCloud Drive	<128KB	–
Box	–	–
SugarSync	Full file	Full file
Seafile	8MB	–
Baidu Netdisk	Full file	Full file

We do not list the web-based case where no deduplication is applied.

tion. The results are also shown in Table 7 (the 3rd column), indicating that only SugarSync and Baidu Netdisk employ cross-user data deduplication.

From the above measurements, we observe that a cloud storage service usually adopts the same data dedup granularity for PC clients and mobile apps, while the web-based data synchronization typically *does not* apply data deduplication. In addition, most services do not support cross-user deduplication at present, perhaps for privacy and security concerns. Consequently, in terms of data deduplication, today's cloud storage services are losing considerable opportunities for optimizing the TUE.

Further, we compare the two types of deduplication granularities to answer this question: *Is the block-level deduplication much better (i.e., has a much larger deduplication ratio) than the full-file deduplication?* Since the computation complexity of block-level deduplication is generally higher than that of full-file deduplication, the answer could help decide whether the block-level deduplication is worthwhile. Note that when referring to the "file blocks," we are dividing files to blocks *in a simple and natural way*, that is to say, by starting from the head of a file with a fixed block size. Thus, clearly, we are not dividing files into blocks in the best possible manner [1, 45], which is much more complicated and computation intensive.

As our collected cloud storage trace (`PC-dataset`, see Section 3.1) in [33] contains both the full-file hash codes and block-level (128KB – 16MB blocks) hash codes of each tracked file, we perform trace-driven simulations to determine both the (cross-user) *dedup ratio* and *dedup time* when each dedup granularity is adopted. Here, the dedup ratio $= \frac{\text{Size of data before deduplication}}{\text{Size of data after deduplication}}$, and the computation complexity mainly comes from client-side MD5 fingerprinting on the PC: B3 @

ALGORITHM 1: Iterative Self Duplication Algorithm

Initialization:
  Set the lower bound: $L = 0$ B, and the upper bound: $U = +\infty$ bytes;
  Guess an initial deduplication, for example, block size: $B_1 = 4$ MB;(According to the measurements of all the existing cloud storage systems we can find, the dedup block size is always 2^n bytes,where n is an integer and $n \in [15, 25]$.Thus, the initial block size B_1 can be reasonably set as 32 KB (2^{15}B), 64 KB, . . . , or 32 MB (2^{25} B),among which 4 MB is the most common.)
Step 1:
  Generate a new compressed file f_1 of B_1 bytes;
  Upload f_1 to the cloud. When f_1 is completely synced, record the total sync traffic: Tr_1;
Step 2:
  Generate another file f_2 by appending f_1 to itself: $f_2 = f_1 + f_1$ (i.e., the "self duplication");
  Upload f_2 to the cloud. When f_2 is completely synced, record the total sync traffic: Tr_2;
Step 3:
if $Tr_2 \ll Tr_1$ **and** Tr_2 is small ($\approx$ tens of KBs) **then**
  B_1 is actually the deduplication block size (B); **exit**;
else there are two cases
  case 1: $Tr_2 < 2B_1$ and Tr_2 is not small (implying that $B_1 > B$) **then**
  Set B_1 as the upper bound: $U \leftarrow B_1$, and decrease the guessing value of B_1: $B_1 \leftarrow \frac{L+U}{2}$;
  case 2: $Tr_2 > 2B_1$ (implying that $B_1 < B$) **then**
  Set B_1 as the lower bound: $L \leftarrow B_1$, and increase the guessing value of B_1: $B_1 \leftarrow \frac{L+U}{2}$;
goto Step 1;

BJ (see Table 2 for its configuration). The simulation results shown in Figure 6 demonstrate that the block-level dedup usually exhibits *trivial superiority* to the full-file dedup in terms of dedup ratio, while incurring *significantly higher* computation overhead as for a relatively small block size ($\leq$ 8MB). Therefore, we have the following implication: For cloud storage providers, in terms of dedup granularity, supporting full-file dedup is basically sufficient to achieve efficient usage of sync traffic in realistic scenarios.

5.2 Peer-Assisted Offloading (P2P)

Experiment 7: As a widely used decentralized content delivery technique exploiting end users' network bandwidths, P2P can minimize cloud-side traffic cost for delivering a popular file shared by multiple users [2, 9, 39]. To uncover the use of P2P in cloud storage services, we let the client download several popular large files (specifically, highly compressed videos) from the cloud and record the actual cloud-side traffic cost. If the cloud-side traffic volume is smaller than the file size, we can confirm the use of P2P.

Our experiment results show that P2P is adopted only by Baidu Netdisk, which has a special design for P2P file sharing. In particular, the P2P data are transported via UDP (rather than TCP) connections to enhance the throughput and parallelism. Quantitatively, higher popularity and larger file size usually imply more traffic savings owing to more data exchange inside the peer swarm. In the extreme case for delivering a highly popular large video file, P2P can significantly cut cloud-side traffic cost by more than 85%. Typically, according to a large-scale measurement study on Dropbox traffic, P2P can considerably cut cloud-side traffic cost by around 25% [20].

On the other hand, it is quite understandable that all of the other services have not adopted P2P yet, probably for privacy and security concerns — they were originally designed for file hosting

Table 8. Average Sync Traffic for 100 Compressed Files' Creations, Where Each File is 1KB

Service	PC client		Web-based		Mobile app	
	Traffic	(*TUE*)	Traffic	(*TUE*)	Traffic	(*TUE*)
Dropbox	182KB	(1.8)	443KB	(4.4)	462KB	(4.6)
OneDrive	419KB	(4.2)	411KB	(4.1)	–	
Google Drive	132KB	(1.3)	779KB	(7.8)	–	
iCloud Drive	861KB	(8.6)	2.2MB	(22)	–	
Box	280KB	(2.8)	1.6MB	(16)	241KB	(2.4)
SugarSync	2.1MB	(21)	549KB	(5.5)	1.0MB	(10)
Seafile	184KB	(1.8)	318KB	(3.2)	–	
Baidu Netdisk	751KB	(7.5)	3.2MB	(32)	1.7MB	(17)

rather than sharing. As content sharing is having an increasing impact on cloud storage bandwidth consumption [20], we suggest that the other services also adopt peer-assisted offloading in the future to achieve cost–benefit trade-offs for both providers and users [21].

6 BATCHING APPROACHES

It is known that syncing a small file incurs poor TUE see Section 4.1) while syncing a number of small files all at once can effectively improve the TUE. In addition to backing up and retrieving files [35], cloud storage services are also widely used for collaboration, such as collaborative document editing, team project building, and database hosting [36]. These collaboration scenarios all involve a special kind of file operation: *frequent edits*, which can be batched through sync deferment.

6.1 File Bundling

Experiment 8: It is reported that the majority of files hosted by cloud storage services are small [15, 23, 33]. Therefore, a number of small files can be bundled into a moderate-sized file to save traffic: in particular, overhead traffic. To determine whether file bundling is adopted by today's cloud storage services, we first generate 100 (distinct) highly compressed files and then move all of them into the sync folder at once. Each file is 1KB in size; thus, TUE = $\frac{\text{Total sync traffic}}{100\text{KB}}$. If file bundling is adopted, the total sync traffic should lie between 100KB and 200KB (the total size of the 100 files plus the overhead traffic) and the TUE should be between 1.0 and 2.0.

The results in Table 8 reveal that Dropbox, Google Drive, and Seafile have definitely adopted file bundling for their PC clients as the TUE is as small as less than 2.0. Further, it is possible that OneDrive and Box have also adopted file bundling for their PC clients because the corresponding sync traffic (419KB and 280KB) is within an order of magnitude of the data update size (100KB). On the contrary, SugarSync has not adopted file bundling yet, for the TUE is as large as 21.

The effect of file bundling for the web-based method is generally weaker than that for PC clients because JavaScript is unable to directly invoke file-level system calls/APIs such as open, close, read, and write [44]. Instead, JavaScript can access users' local files only in an indirect and constrained manner, thus bringing about more overhead traffic. Still worse, file bundling is not used by the mobile apps of OneDrive, Google Drive, iCloud Drive, and Seafile simply because they do not offer an operational user interface (UI) via which a user can move a bundle of files into the sync folder all at once. In fact, implementing such a UI is not difficult for mobile app developers in practice, since there are off-the-shelf APIs in both Android and iOS Software Development Kit (SDKs).

6.2 Sync Deferment

Frequent edits imply that a file is edited in a frequent and incremental manner. Thus, they exhibit diverse data update patterns in terms of data update size and rate. The `Dropbox-dataset` (see Section 3.1) reveals that for 8.5% of Dropbox users, more than 10% of their sync traffic is caused by frequent edits [36]. Further, frequent edits may well incur abundant overhead traffic that far exceeds the amount of useful data update traffic sent by the user client over time, which is referred to as the *traffic overuse problem* [36]. To overcome this problem, some cloud storage services intentionally defer the sync process for a certain period of time to batch the file updates.

Experiment 9: To check the use of sync deferment, we conduct the "X KB/X sec" appending experiments—we append X random kilobytes to an empty file inside the sync folder every X seconds, until the total appended bytes reach a certain size C (typically C = 1 MB). We use random bytes since they are difficult to compress, thus preventing file compression from influencing our measurements of TUE. All of the experiments are performed using S1 @ SF (see Table 2) with 20Mbps of bandwidth, and the RTT latency (between S1 and each cloud) is between 31msec and 77msec. In terms of service access method, we examine the PC client only because almost all frequent edits are generated from PC clients in practice[4]. Experiments with other benchmarks with different network environments and hardware configurations will be presented in Section 6.3.

Our goal is threefol by doing the experiments: (1) we observe and understand the sync traffic and TUE in response to frequent edits; (2) we aim to discover whether the cloud storage service has used the *sync deferment* in order to avoid or mitigate the traffic overuse problem; and 3) if the sync deferment is adopted, we want to measure how long the sync deferment is.

First, to investigate the impact of frequent file edits on the TUE, we examine the cases for $X \in \{1, 2, \ldots, 10\}$. As depicted in Figure 7, the seven services (excluding Baidu Netdisk) exhibit diverse and interesting phenomena. First, frequent edits to a file often lead to large TUE (the aforementioned traffic overuse problem); the maximum TUE can reach 6.4, 570, 90, 93, 234, 90, and 190, respectively. Second, we observe (except in the sync deferment cases: Figure 7(d), 7(f), and 7(g)) that TUE generally decreases as the edit frequency (= $\frac{1024}{X}$) decreases. The reason is straightforward: though the total data update size is always C = 1MB, a lower data update frequency implies fewer data sync events and, thus, the overhead traffic is reduced.

A detailed, yet critical, question is this: *Why are the maximum TUEvalues of OneDrive (570), Google Drive (90), iCloud Drive (93), Box (234), SugarSync (90), and Seafile (190) much larger than that of Dropbox (6.4)?* The answer can be found from their data sync mechanisms and granularities (see Section 4.3). As OneDrive, Google Drive, and Box employ *full-file sync*, their maximum TUE values are naturally large. In contrast, as Dropbox employs chunk-level *delta sync* and the chunk size is quite small, its maximum TUE value is naturally small. Furthermore, the maximum TUE of Seafile greatly exceeds that of Dropbox because the chunk size of Seafile is variable and usually much larger than that of Dropbox, as illustrated in Figure 4(a).

The situation of iCloud Drive and SugarSync is a bit complicated: when we perform a single file edit, they use fine-grained delta sync to save traffic, as demonstrated in Figure 4(a); in contrast, when we perform frequent file edits, they both degrade to coarse-grained full-file sync, which results in enormous traffic cost. To demystify this seemingly weird phenomenon, we measure the CPU use of the PC client of each cloud storage service on handling a single 10KB appending and the 10KB/10sec appending experiments. Our measurement results, as listed in Table 9, indicate that the PC client of iCloud Drive and SugarSync requires higher CPU use on handling a single file edit than frequent file edits. Hence, we infer that iCloud Drive and SugarSync are trading

[4]Also, the UIs of web browsers and mobile apps are usually not fit or not convenient for performing frequent edits to a file.

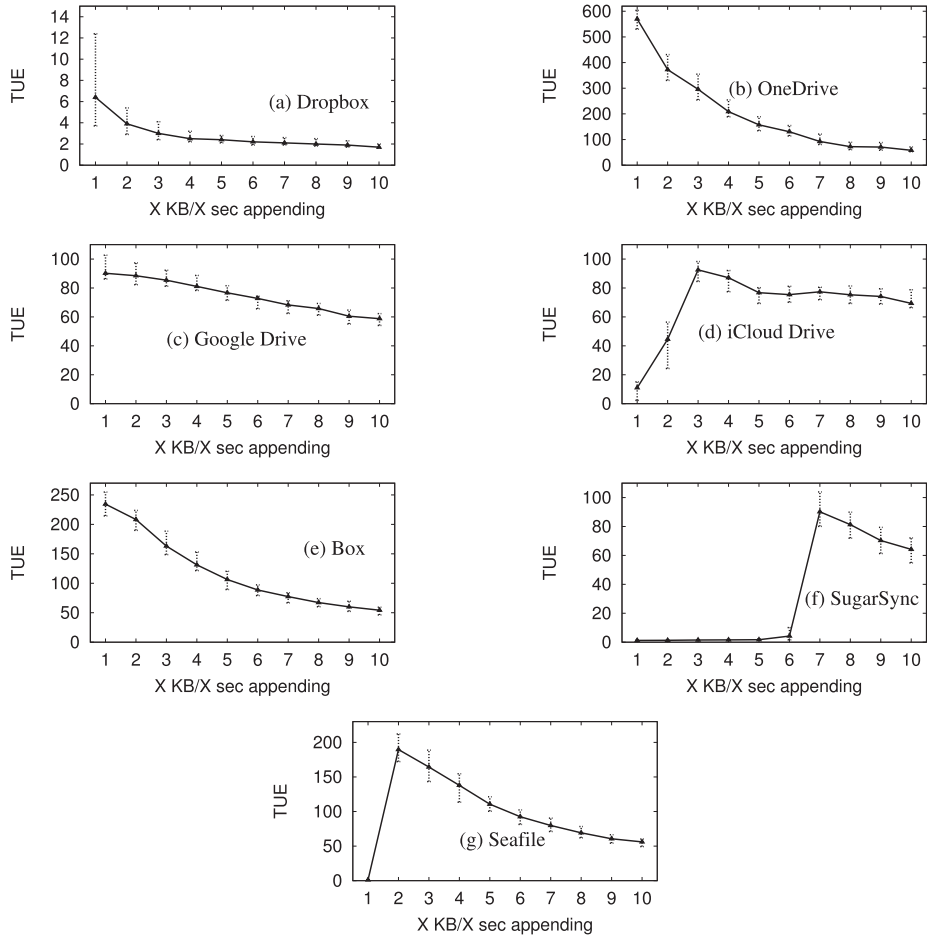

Fig. 7. TUE of seven cloud storage services in response to controlled frequent file edits. The average RTT latency between the PC client (at San Francisco) and the cloud server is 62, 31, 77, 43, 42, 71, and 42msec, respectively. Baidu Netdisk is not included since its PC client does not support frequent edits to a file.

Table 9. Average CPU Utilization (On One Core) of the PC Client of Each Cloud Storage Service on Handling a Single 10KB Appending and the 10KB/10sec Appending Experiments

Service	**CPU Utilization for a single 10KB appending**	**CPU Utilization for 10KB/10 sec appending**
Dropbox	26.6%	26.6%
OneDrive	6.6%	6.6%
Google Drive	8.2%	8.2%
iCloud Drive	**16%**	**9%**
Box	22.6%	22.6%
SugarSync	**24.6%**	**12%**
Seafile	4.6%	4.6%

The bold type denotes the cases when the CPU utilization for a single 10KB appending differs from the CPU utilization for 10KB/10 sec appending, which thus implies a serviceâ..s trading increased network traffic cost for reduced computation cost in the presence of frequent file edits.

increased network traffic cost for reduced computation cost in the presence of frequent file edits. In contrast, for Dropbox, OneDrive, Google Drive, Box, and Seafile, their PC client exhibits the same CPU use on handling a single file edit or frequent file edits because they use the same data sync mechanism to deal with both edit cases. In addition, Table 9 confirms that delta sync actually incurs considerably higher CPU use than full-file sync.

On the other hand, there are cases (in Figure 7(d) for $X = 1$, Figure 7(f) for $X \leq 6$, and Figure 7(g) for $X = 1$) where the TUE is very small (close to 1.0). According to our observations, 7(d) iCloud Drive, 7(f) SugarSync, and 7(g) Seafile deal with traffic overuse by batching file updates using a *fixed* sync deferment: T seconds (which cannot be reconfigured by users). In Figure 7(d), 7(f), and 7(g), there is a "spike" of TUE in each figure, indicating the rough range of T. Thus, we infer that $T_{iCloudDrive} \in (1, 3)$ sec, $T_{SugarSync} \in (5, 7)$ sec, and $T_{Seafile} \in (1, 3)$ sec. Moreover, to determine a more accurate value of T, we further tune X from integers to floats. Specifically, we experiment with $X = 1.1, 1.2, \ldots, 2.9$ for iCloud Drive, $X = 5.1, 5.2, \ldots, 6.9$ for SugarSync, and $X = 1.1, 1.2, \ldots, 2.9$ for Seafile and then find that $T_{iCloudDrive} \approx 2.0$sec, $T_{SugarSync} \approx 6.0$sec, and $T_{Seafile} \approx 2.0$sec.

Sync Deferment, Byte Counter, or Update Counter? One may have the following doubt: *Is it possible that the deferred data synchronization of 7(d) iCloud Drive, 7(f) SugarSync, and 7(g) Seafile is triggered by a byte counter or an update counter rather than the time threshold (T)?* In other words, the three concerned services may trigger the data synchronization once the *number of uncommitted bytes* (byte counter) or the *number of uncommitted data updates/operations* (update counter) exceeds a certain value. This doubt can be addressed in two cases:

Case 1: If the data synchronization is triggered by a *byte counter*, the resulting *TUE* would be close to 1.0 for all values of X according to our previous study on the byte counter–based "efficient batched synchronization" [36]. This is clearly not true, as illustrated by Figures 7(d), 7(f), and 7(g).
Case 2: If the data synchronization is triggered by an *update counter*, the resulting TUE in Figures 7(d), 7(f), and 7(g) would *linearly* decrease as the edit period (X sec) increases. This is not true, either.

Therefore, we conclude that the deferred data synchronization is not triggered by a byte counter or an update counter but rather by a certain period of time.

Limitation of Fixed Sync Deferments. Unfortunately, fixed sync deferments currently employed by iCloud Drive, SugarSync, and Seafile are considerably limited in terms of usage scenarios. As shown in Figures 7(d), 7(f), and 7(g), the traffic overuse problem still occurs when $X > T$.

To overcome the limitation of fixed sync deferments, we propose an ASD mechanism. ASD adaptively tunes its sync deferment (T_i) to follow the latest (say, the ith) data update. In other words, when data updates occur more frequently, T_i gets shorter; when data updates occur less frequently, T_i gets longer. In either case, T_i tends to be slightly longer than the latest inter-update time so that frequent edits can be properly batched for synchronization (without harming user experience). Specifically, T_i can be adapted in such an iterative manner:

$$T_i = \min\left(\frac{T_{i-1}}{2} + \frac{\Delta t_i}{2} + \epsilon,\ T_{max}\right) \tag{2}$$

where Δt_i is the inter-update time between the (i-1)th and the ith data updates, and $\epsilon \in (0, 1.0)$ is a small constant that guarantees T_i to be slightly longer than Δt_i in a small number of iterations. T_{max} is also a constant representing the upper bound of T_i, as a too large T_i will harm user experience by bringing about intolerably long sync delay. If iCloud Drive used ASD on handling the "X KB/X

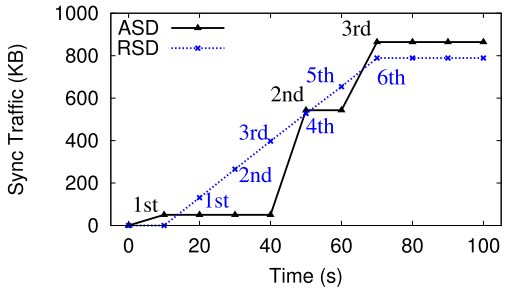

Fig. 8. The performance (in terms of sync time) and overhead (in terms of sync traffic) of ASD and RSD in a typical benchmark experiment. During 100sec, ASD performs 3 sync operations using 880KB of traffic, while RSD performs 6 sync operations using 790KB of traffic.

sec" ($X > T_{iCloudDrive}$) appending experiments, the resulting TUE will be close to 1.0 rather than the original 70+ for $X \in \{5, 6, 7, 8, 9, 10\}$. The situation is similar for SugarSync and Seafile.

Further, we note that although ASD is able to minimize sync traffic, it does not always perform well in terms of sync time (and thus may impair the user experience). To address the issue, Sáiz-Laudó et al. propose an enhanced, more fine-grained mechanism called *rate-based sync deferment* (RSD) to reduce the file sync time of ASD [51]. Seeking a quantitative comparison between ASD and RSD, we carry out a benchmark experiment using a real desktop client (OneDrive version 18.025). The results in Figure 8 show that RSD can actually reduce the sync time per operation by around 2× compared with ASD, while bringing a comparable amount of sync traffic.

6.3 Impact of Network and Hardware

Experiment 10, Network Environment: To study the impact of network environment (including both bandwidth and latency) on the TUE, we conduct the following two batches of experiments. The first batch of experiments is performed on B1 @ BJ. It represents a relatively poor network environment: low bandwidth (nearly 1.6Mbps) and long latency (between 136msec and 299msec) relative to the cloud server, because the studied cloud storage services (except Baidu Netdisk) are mainly deployed in the United States. After repeating Experiments 1 through 9 in this network environment, we compare the results with the corresponding results by using S1 @ SF with abundant bandwidth (nearly 20Mbps) and short latency (between 31msec and 77msec), which represents a fine network environment. The second batch of experiments is performed by using S1 @ SF with controlled bandwidth (between 1.6Mbps and 20Mbps) and latency (between 40msec and 1000msec) so that we are able to get fine-grained results about how the network environment impacts the TUE.

From the two batches of experiments, we mainly get the following findings and implications:

- The TUE of a simple file operation is usually not affected by network environment.
- However, in the case of frequent file edits, a user client with relatively low bandwidth or long latency can usually save more sync traffic.

Specifically, for the first batch of experiments, we plot the TUE of seven cloud storage services on handling the "X KB/X sec" appending experiment in San Francisco and Beijing in Figure 9, respectively. In each subfigure, the two curves ("@ SF" *vs.* "@ BJ") clearly illustrate that poor network environment leads to smaller TUE, especially when the edit period (X sec) is short (excluding the sync deferment cases). For the second batch of experiments, as a typical example, we plot the TUE of OneDrive on handling the "1KB/sec" appending experiment with variable bandwidths and

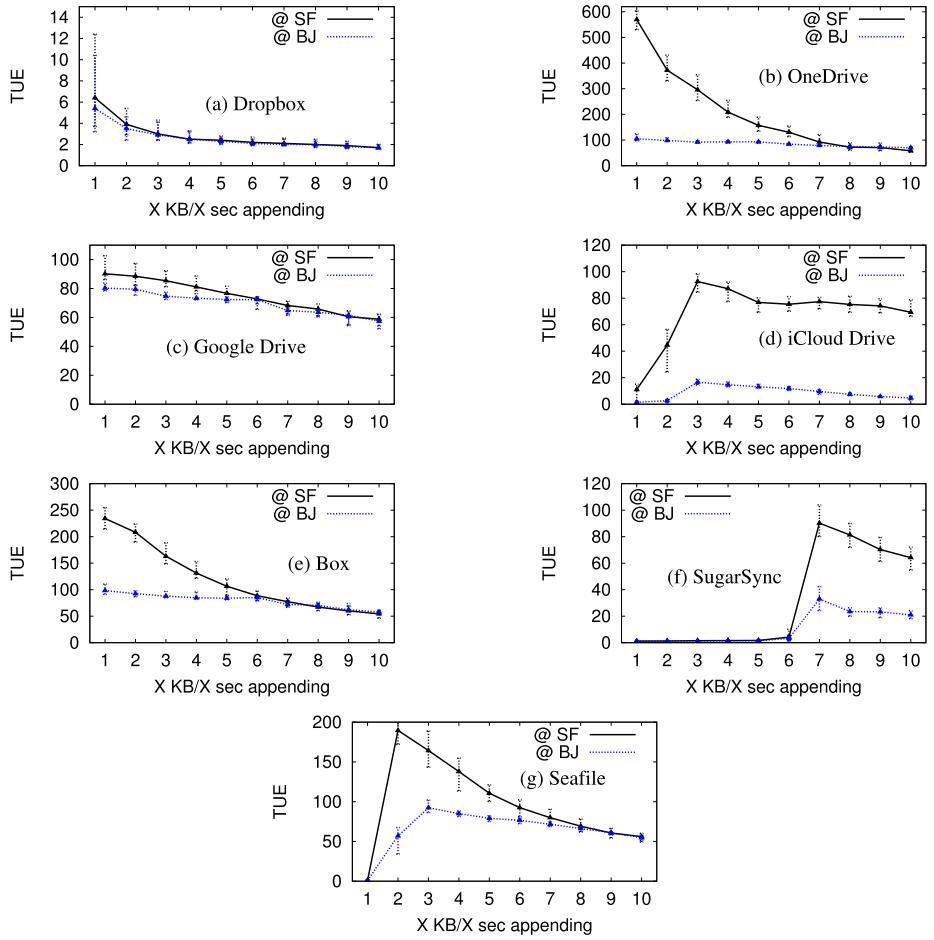

Fig. 9. TUE of seven cloud storage services in response to controlled frequent file edits. The average RTT latency between the PC client at Beijing and the cloud server is 200, 227, 248, 136, 162, 233, and 299msec, respectively. Baidu Netdisk is not included since its PC client does not support frequent edits to a file.

latencies in Figure 10(a) and Figure 10(b), respectively. In Figure 10(a), the latency is fixed to around 40msec and the bandwidth is tuned from 1.6Mbps to 20Mbps. In Figure 10(b), the bandwidth is fixed to around 20Mbps and the latency is tuned from 40msec to 1000msec. The two figures illustrate that higher bandwidth or shorter latency leads to larger TUE for OneDrive. In fact, this phenomenon happens to all of our studied cloud storage services.

Experiment 11, Hardware Configuration: Next, we examine the impact of hardware configuration on TUE by repeating Experiments 1 through 9 with distinct client machines: S1 (a typical current machine), S2 (an outdated machine), and S3 (an advanced machine). Their detailed hardware information is listed in Table 2. All of the experiments are performed in SF with abundant bandwidth (nearly 20Mbps) and short latency (between 31msec and 77msec).

Through the results of Experiment 11, we observe that the TUE of a simple file operation generally has no relation with hardware configuration, but the TUE of frequent file edits is actually affected by hardware configuration. As a typical example, in Figure 10(c) we plot the TUE of

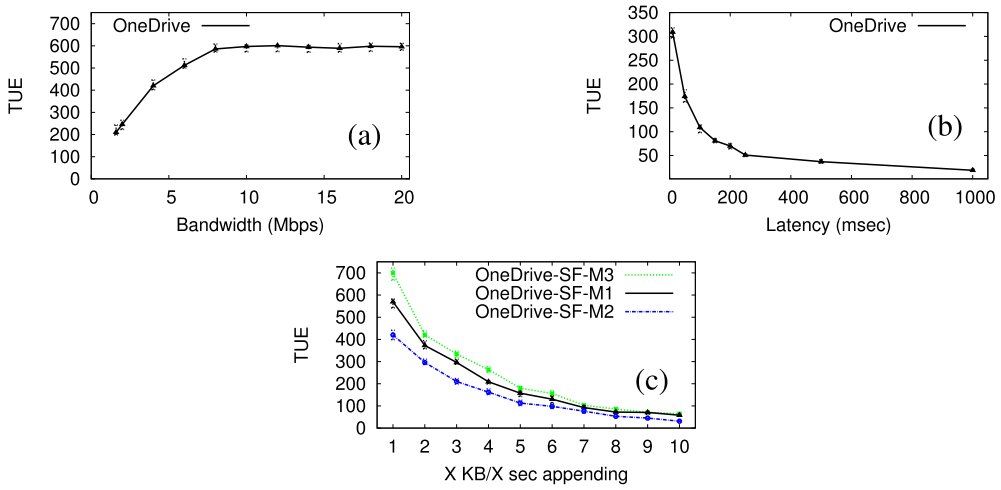

Fig. 10. TUE of OneDrive on handling the (a) "1KB/sec" appending experiment with variable bandwidths, (b) "1KB/sec" appending experiment with variable latencies, and (c) "X KB/X sec" appending experiment with distinct hardware configurations.

OneDrive on handling the "X KB/X sec" appending experiment with S1, S2, and S3. The three curves clearly demonstrate that slower hardware incurs less sync traffic.

Why Do Network Environment and Hardware Configuration Impact TUE? To explore the reason why network environment and hardware configuration impact the TUE, we analyze the communication principles of data synchronization for the studied cloud storage services by reverse engineering their data sync processes (see Section 3.2). The analysis reveals that in the presence of frequent edits to a file, the user client does *not* always synchronize every file edit to the cloud separately. Instead, the user client often naturally (or unintentionally) batches multiple file edits for data synchronization. Specifically, a new file edit (or a sequence of new file edits) is synchronized to the cloud server(s) when at least the following two conditions are both satisfied:

Condition 1: The previous file edit (or batch of file edits) has been completely synced to the cloud.
Condition 2: The client machine has finished calculating the latest metadata of the edited file.

Regarding Condition 1, when the network environment is relatively poor, synchronizing the previous file edit (or the previous batch of file edits) takes more time; thus, the client needs to wait for a longer period of time to synchronize the new file edit. Regarding Condition 2, when the client runs on top of slower hardware, calculating the latest metadata (mostly the rolling checksums of the updated file; see Section 4.3, which is computation intensive) also requires a longer period of time. Because the failure of either condition will cause the new file edit (or the sequence of new file edits) to be naturally batched, poor network environment or poor hardware increases the probability that a file edit gets batched and thereby optimizes the TUE.

Finally, combining all of the findings in this section, we get the following implication. In the case of frequent file edits, today's cloud storage services actually bring good news (in terms of TUE) to those users with relatively poor hardware or Internet access.

6.4 General Insights with Respect to Frequent Edits

In Sections 6.2 and 6.3, we ran several benchmark experiments with respect to frequent edits. Below, we briefly summarize our observations and insights obtained from the measurement results.

Table 10. Thumbnail View Traffic and Support for JPEG Images, MPEG-4 Videos, and Microsoft Word Docs

Service	Thumbnail view traffic	JPEG	MPEG-4	Microsoft Word
Dropbox	66KB	Yes	Yes	–
OneDrive	67KB	Yes	Yes	Yes
Google Drive	44KB	Yes	Yes	Yes
iCloud Drive	978KB	Yes	–	–
Box	362KB	Yes	Yes	–
SugarSync	23KB	Yes	–	–
Seafile	50KB	Yes	–	–
Baidu Netdisk	17KB	Yes	Yes	–

- The traffic overuse problem occurs when there are numerous small edits to files that occur at intervals on the order of several seconds. Under these situations, cloud storage applications can hardly batch updates together, causing the amount of sync traffic to be several orders of magnitude larger than the actual size of the file.
- In general, cloud storage applications synchronize data to the cloud only after the local data has been indexed and prior synchronizations have been successfully resolved. File edits that occur within relatively small intervals are likely to be batched owing to file indexing. By exploiting this principle, some cloud storage services actively batch the edits using a fixed sync deferment to significantly reduce sync traffic costs.
- The traffic overuse problem is actually made worse by faster network connections and higher disk speeds. The former make the success acknowledgment of each synchronization return in shorter time and the latter accelerate the calculation of file indexes.

7 WEB-SPECIFIC APPROACHES

Among the three access methods to cloud storage services, the web-based method is usually the worst in terms of efficiency and functionality, as shown in previous sections. Nevertheless, it is the the most pervasive and OS independent since all the major cloud storage services support web-based access, while providing only PC clients and mobile apps for a limited set of OS distributions and devices. In this section, we focus on the traffic-saving approaches specific to web-based access.

7.1 Thumbnail Views

Experiment 12: Thumbnail views are now used by many Internet services to avoid loading the real content of files when the user is accessing the files via a web browser. A thumbnail view is typically only severalkilobytess in size, presenting a sketch for an image, a video, or a document so that the user can get a brief understanding of the file at a glance. Note that a thumbnail view is more complex than an icon, which indicates only the type of a file. Detecting the use of thumbnail views is quite straightforward—we just need to "open" a folder that includes a few files via the web-based method and then observe the resulting sync traffic.

For each cloud storage service, we use the Chrome web browser to open a folder that contains one JPEG image of 1MB, one Microsoft Word document of 5MB, and one MPEG-4 video of 10MB. From the experiment results listed in Table 10, we observe that all eight cloud storage services have used thumbnail views for certain types of files, and the resulting sync traffic (for delivering the thumbnail views) stays between tens and hundreds of kilobytes, which are much smaller than the total size of the three files (16MB). Specifically, all services support thumbnail views for JPEG

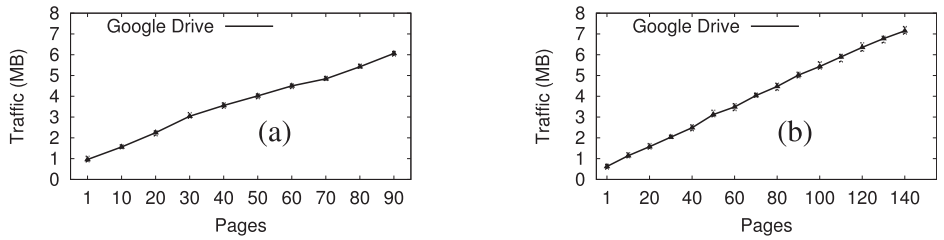

Fig. 11. Network traffic use as we gradually view (a) a 3.5MB Microsoft Word document that includes 93 pages and (b) a 4MB PDF file that includes 140 pages, via the Chrome web browser.

images, over half of the services support thumbnail views for MPEG-4 videos, but only OneDrive and Google Drive support thumbnail views for Microsoft Word documents. This is because generating thumbnail views for Microsoft Word documents is more complicated than generating thumbnail views for JPEG images and MPEG-4 videos. Of course, this complication can be effectively addressed by calculating the thumbnail views offline on storage servers.

7.2 Dynamic Content Loading (DCL)

Experiment 13: Currently, many cloud storage services provide users with web-based online content viewing and editing for certain types of files, especially documents. In contrast to thumbnail views, which present only a simplified sketch, DCL presents an on-demand part of the *actual* content of a file to the user. The heuristic behind DCL is intuitive: the user usually intends to view and/or edit only a part of the file, thus, it is unnecessary to load the entire content of the file. Detecting the use of DCL is also straightforward—we just need to "open" a multi-page document via the web-based method and then observe the resulting sync traffic. Based on the sync traffic and document size, we can even roughly estimate how much content is loaded from the file.

Our measurement results show that DCL is currently used only by Google Drive to avoid loading the full content of a document. Typically, only one page is loaded at first, and the remaining pages are dynamically loaded according to user activity. For example, when we use the Chrome web browser to open a 3.5MB Microsoft Word document that includes 93 pages in Google Drive, the resulting traffic use is only 920KB at first. Later, when we gradually view the remaining pages at a normal reading speed, the traffic use linearly increases as depicted in Figure 11(a). Finally, when we have viewed all pages, the traffic use just reaches 6MB, slightly exceeding the size of the document. Additionally, DCL is also applied to Microsoft PowerPoint documents, Microsoft Excel documents, PDF files (as shown in Figure 11(b)), and so on. Given that DCL can make traffic use almost proportional to the pages viewed, we suggest that DCL should be adopted by the other services that host numerous documents.

8 RELATED WORK

As cloud storage services are becoming more pervasive and changing the way people store and share data, a number of research efforts have been made in both academia and industry, including the design and implementation of service infrastructure [8, 13, 23, 39, 40, 49, 58, 59, 61, 67], integration services with various features and functionalities [10, 24, 25, 29, 34, 36, 52, 54, 66], and performance measurement [5, 14, 15, 31, 60, 62]. In this article, we concentrate on the data sync mechanisms that are fundamental to the performance of cloud storage services, together with the resulting sync traffic use and efficiency.

Dropbox. Dropbox is one of the earliest and most popular cloud storage services; its data sync mechanism has been studied in depth in [15, 36]. Through an ISP-level large-scale measurement,

Drago et al. first uncovered the performance bottlenecks of Dropbox due to both the system architecture and the data sync mechanism [15]. They suggest a bundling sync scheme with delayed sync ACK to improve the sync performance of Dropbox. Gonçalves et al. propose a hierarchical two-layer model for representing Dropbox client behavior and workload patterns [19]. Li et al. identify a pathological issue that may lead to the "traffic overuse problem" in Dropbox by uploading a large amount of unnecessary (overhead) traffic [36]. They devise an efficient batched sync algorithm (named UDS) to address this issue. Lopez et al. put forward two challenging issues related to the design of Dropbox: fine-grained programmable elasticity and efficient change notification to millions of users [40]. They propose and implement a novel data sync architecture (named StackSync) to solve the two issues.

Other Cloud Storage Services. Quite a few measurement studies have covered the traffic use of other cloud storage services (beyond Dropbox). Hu et al. examine "the good, the bad and the ugly" of four cloud storage services by comparing their traffic use, delay time, and CPU use of uploading new files [26]. They observe that sync traffic use varies substantially with factors such as file size, data compressibility, and data duplication levels. Paying attention to some unique quality-of-service facets, Gracia-Tinedo et al. conduct an active measurement study on Dropbox, Box, and SugarSync through their RESTful APIs [22]. Drago et al. further compare the system capabilities of five cloud storage services and find that each service has limitations with regard to data synchronization [14]. Li et al. conduct a comparative study of various data sync mechanisms used by a number of popular cloud storage services, unravel the pathological processes for their traffic overuse problems, and point out that coarse-grained data sync mechanism design may lead to severe traffic overuse [37]. Gracia-Tinedo et al. present the internal structure and a massive measurement study of UbuntuOne, paying special attention to the behavior of users that should be exploited to optimize the performance of data sync operations [23]. All of these studies confirm the importance of sync traffic use and the possibility of optimizing sync traffic use in an appropriate manner.

Mobile Cloud Storage Services. Owing to the propagation of mobile devices, considerable attention has been paid to mobile cloud storage services. By analyzing a large-scale sync request dataset from a commercial mobile cloud storage service, Li et al. find that mobile cloud storage service is dominated by file uploads, thus suggesting that incremental sync and chunk-level deduplication can be reasonably omitted in mobile scenarios [35]. In addition, Bai and Zhang note that the inefficiency of today's mobile cloud storage services stems from the inability of cloud storage services to monitor and service file operations on mobile devices as well as the loose coupling between mobile apps and storage servers [4]. Based on this insight, they design the StoArranger framework to coordinate, rearrange, and transform cloud storage communications on mobile devices. Ultimately, throughout this article, we did not discover substantial differences in network-level efficiency between WiFi and 4G access methods, implying that the 4G access method (whose traffic is much more expensive than WiFi traffic) for cloud storage services still deserves extensive optimization.

Comparison with Our Work. While previous work covers the data sync mechanism as one of the key operations and resulting traffic use, none tries to understand the *efficiency* of traffic usage *quantitatively* and *comprehensively*. Due to system complexity and implementation difference, one can hardly form a general and unified view of traffic use efficiency, not to mention further improvement. Our work[5] is different from and complementary to previous studies by quantifying

[5]This article extends and improves on its preliminary version [33] (an excerpt also appeared in [32]) in terms of *studied services*, *research methodology*, *measurement experiments*, *target cloud storage techniques*, and *novel results and findings*. In

and optimizing TUE, the pivotal network-level efficiency for cloud storage services. Based on the measurements and analysis of eight state-of-the-art cloud storage services, we unravel the key impact factors and design choices that may significantly affect TUE. Specifically, we consider the diversity of access methods, client locations, hardware configurations, and network conditions to match real-world use. Indeed, we discover that these factors lead to different traffic use patterns, some of which are not expected. Most importantly, we provide easy-to-follow guidance and implications for both service providers and end users to economize their sync traffic use.

9 CONCLUSION

The tremendous increase in data sync traffic has brought growing pains to today's cloud storage services in terms of both costs and performance penalties on both server and client sides. Driven by this industry-wide problem, this article quantifies and analyzes the data sync TUE of eight widely used cloud storage services by conducting comprehensive benchmark experiments and reverse engineering their data sync processes. In particular, we uncover their nine-fold practical endeavors for optimizing the TUE, which are further classified into four types of approaches (intra-file, cross-file/-user, batching, and web-specific).

Our results and findings illustrate that despite the manifold TUE optimization efforts, much of the data sync traffic consumed by cloud storage services is unnecessary and can be effectively avoided or mitigated by careful design and implementation of data sync mechanisms. There is still enormous space for optimizing the network-level efficiency of cloud storage services. We hope that our work can stimulate cloud storage designers to enhance their system and software, while guiding users to pick appropriate services currently available.

REFERENCES

[1] Bhavish Agarwal, Aditya Akella, Ashok Anand, Athula Balachandran, Pushkar Chitnis, Chitra Muthukrishnan, Ramachandran Ramjee, and George Varghese. 2010. EndRE: An end-system redundancy elimination service for enterprises. In *Proceedings of NSDI*. USENIX, 419–432.

[2] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. 2004. A survey of peer-to-peer content distribution technologies. *Computer Surveys* 36, 4 (2004), 335–371.

[3] Amazon Web Services (AWS). 2017. Amazon S3 pricing policy. Retrieved December 10, 2018 from https://aws.amazon.com/s3/pricing.

[4] Yongshu Bai and Yifan Zhang. 2017. StoArranger: Enabling efficient usage of cloud storage services on mobile devices. In *Proceedings of ICDCS*. IEEE, 1476–1487.

[5] Andreas Bergen, Yvonne Coady, and Rick McGeer. 2011. Client bandwidth: The forgotten metric of online storage providers. In *Proceedings of PacRim*. IEEE, 543–548.

[6] Enrico Bocchi, Idilio Drago, and Marco Mellia. 2015. Personal cloud storage benchmarks and comparison. *IEEE Transactions on Cloud Computing* 5, 4 (2015), 751–764.

[7] David Bolen. 2001. A question about the default chunk size of rsync. Retrieved December 10, 2018 from http://lists.samba.org/archive/rsync/2001-November/000595.html.

[8] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, et al. 2011. Windows Azure storage: A highly available cloud storage service with strong consistency. In *Proceedings of SOSP*. ACM, 143–157.

[9] Guihai Chen and Zhenhua Li. 2007. Peer-to-peer network: Structure, application and design. *Tsinghua University Press* (2007), 1–337.

[10] Yanpei Chen, Kiran Srinivasan, Garth Goodson, and Randy Katz. 2011. Design implications for enterprise storage systems via multi-dimensional trace analysis. In *Proceedings of SOSP*. ACM, 43–56.

[11] James Cowling. 2016. Inside the Magic Pocket. Retrieved December 10, 2018 from http://blogs.dropbox.com/tech/2016/05/inside-the-magic-pocket.

[12] China Software Development Network (CSDN). 2016. Baidu Netdisk attracted 400 million users in its first 4 years (in Chinese). Retrieved December 10, 2018 from http://www.csdn.net/article/a/2016-10-12/283196.

brief, it provides an up-to-date, comprehensive, and in-depth view on the network-level efficiency of today's cloud storage services.

[13] Yong Cui, Zeqi Lai, Xin Wang, Ningwei Dai, and Congcong Miao. 2015. QuickSync: Improving synchronization efficiency for mobile cloud storage services. In *Proceedings of MobiCom*. ACM, 592–603.
[14] Idilio Drago, Enrico Bocchi, Marco Mellia, Herman Slatman, and Aiko Pras. 2013. Benchmarking personal cloud storage. In *Proceedings of IMC*. ACM, 205–212.
[15] Idilio Drago, Marco Mellia, Maurizio M. Munafo, Anna Sperotto, Ramin Sadre, and Aiko Pras. 2012. Inside Dropbox: Understanding personal cloud storage services. In *Proceedings of IMC*. ACM, 481–494.
[16] Idilio Drago, Marco Mellia, Maurizio M. Munafo, Anna Sperotto, Ramin Sadre, and Aiko Pras. 2012. Large-scale Dropbox traces collected at the ISP level. Retrieved December 10, 2018 from https://www.simpleweb.org/wiki/index.php/Traces#Dropbox_Traffic_Traces.
[17] Roy T. Fielding and Richard N. Taylor. 2000. *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. Dissertation. University of California, Irvine, Irvine, CA.
[18] Galder Zamarreno. 2010. Why RESTful Design for Cloud is Best. Retrieved December 10, 2018 from https://www.slideshare.net/galderz/restful-designjbw-summit2010.
[19] Glauber Gonçalves, Idilio Drago, Ana Paula Couto Da Silva, Alex Borges Vieira, and Jussara M. Almeida. 2014. Modeling the Dropbox client behavior. In *Proceedings of ICC*. IEEE, 1332–1337.
[20] Glauber Gonçalves, Idilio Drago, Ana Paula Couto Da Silva, Alex Borges Vieira, and Jussara M. Almeida. 2016. The impact of content sharing on cloud storage bandwidth consumption. *IEEE Internet Computing* 20, 4 (2016), 26–35.
[21] Glauber Gonçalves, Alex Borges Vieira, Idilio Drago, Ana Paula Couto Da Silva, and Jussara M. Almeida. 2017. Cost-benefit tradeoffs of content sharing in personal cloud storage. In *Proceedings of MASCOTS*. IEEE, 32–42.
[22] Raul Gracia-Tinedo, Marc Sanchez Artigas, Adrian Moreno-Martinez, Cristian Cotes, and Pedro Garcia Lopez. 2013. Actively measuring personal cloud storage. In *Proceedings of IEEE CLOUD*. 301–308.
[23] Raúl Gracia-Tinedo, Yongchao Tian, Josep Sampé, Hamza Harkous, John Lenton, Pedro García-López, Marc Sánchez-Artigas, and Marko Vukolic. 2015. Dissecting UbuntuOne: Autopsy of a global-scale personal cloud back-end. In *PProceedings of IMC*. ACM, 155–168.
[24] Danny Harnik, Ronen I. Kat, Oded Margalit, Dmitry Sotnikov, and Avishay Traeger. 2013. To zip or not to zip: Effective resource usage for real-time compression. In *Proceedings of FAST*. USENIX, 229–242.
[25] Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. 2010. Side channels in cloud services: Deduplication in cloud storage. *IEEE Security & Privacy* 8, 6 (2010), 40–47.
[26] Wenjin Hu, Tao Yang, and Jeanna N. Matthews. 2010. The good, the bad and the ugly of consumer cloud storage. *ACM SIGOPS Operating Systems Review* 44, 3 (2010), 110–115.
[27] Matthew Humphries. 2017. Google Drive Passes 2 Trillion Files Stored (and 800 Million Users). Retrieved December 10, 2018 from http://www.pcmag.com/news/353541/google-drive-passes-2-trillion-files-stored.
[28] Mathew Ingram. 2011. What Happens When the Cloud Meets a Bandwidth Cap. Retrieved December 10, 2018 from http://gigaom.com/2011/05/04/what-happens-when-the-cloud-meets-a-bandwidth-cap.
[29] E. Jinlong, Yong Cui, Peng Wang, Zhenhua Li, and Chaokun Zhang. 2017. CoCloud: Enabling efficient cross-cloud file collaboration based on inefficient web APIs. In *Proceedings of INFOCOM*. IEEE, 1–9.
[30] Dhiru Kholia and Przemyslaw Wegrzyn. 2013. Looking inside the (drop) box. In *Proceedings of the 7th USENIX Workshop on Offensive Technologies (WOOT'13)*. USENIX, 1–7.
[31] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. 2010. CloudCmp: Comparing public cloud providers. In *Proceedings of IMC*. ACM, 1–14.
[32] Zhenhua Li, Yafei Dai, Guihai Chen, and Yunhao Liu. 2016. Toward network-level efficiency for cloud storage services. In *Chapter 8 of Content Distribution for Mobile Internet: A Cloud-based Approach*. Springer Science+Business Media Singapore, 167–196.
[33] Zhenhua Li, Cheng Jin, Tianyin Xu, Christo Wilson, Yao Liu, Linsong Cheng, Yunhao Liu, Yafei Dai, and Zhi-Li Zhang. 2014. Towards network-level efficiency for cloud storage services. In *Proceedings of IMC*. ACM, 115–128.
[34] Zhenhua Li and Jian Li. 2014. Deficiency of scientific research behind the price war of cloud storage services. *Communications of China Computer Federation* 10, 8 (2014), 36–41.
[35] Zhenyu Li, Xiaohui Wang, Ningjing Huang, Mohamed Ali Kaafar, Zhenhua Li, Jianer Zhou, Gaogang Xie, and Peter Steenkiste. 2016. An empirical analysis of a large-scale mobile cloud storage service. In *PProceedings of IMC*. ACM, 287–301.
[36] Zhenhua Li, Christo Wilson, Zhefu Jiang, Yao Liu, Ben Y. Zhao, Cheng Jin, Zhi-Li Zhang, and Yafei Dai. 2013. Efficient batched synchronization in Dropbox-like cloud storage services. In *Proceedings of Middleware*. ACM/IFIP/USENIX, 307–327.
[37] Zhenhua Li, Zhi-Li Zhang, and Yafei Dai. 2013. Coarse-grained cloud synchronization mechanism design may lead to severe traffic overuse. *Journal of Tsinghua Science and Technology* 18, 3 (2013), 286–297.
[38] Xing Lin, Guanlin Lu, Fred Douglis, Philip Shilane, and Grant Wallace. 2014. Migratory compression: Coarse-grained data reordering to improve compressibility. In *Proceedings of FAST*. USENIX, 257–271.

[39] Fangming Liu, Ye Sun, Bo Li, Baochun Li, and Xinyan Zhang. 2010. FS2You: Peer-assisted semi-persistent online hosting at a large scale. *IEEE Transactions on Parallel and Distributed Systems* 21, 10 (2010), 1442–1457.
[40] Pedro Garcia Lopez, Marc Sanchez-Artigas, Sergi Toda, Cristian Cotes, and John Lenton. 2014. StackSync: Bringing elasticity to Dropbox-like file synchronization. In *Proceedings of Middleware*. ACM, 49–60.
[41] Beijing Haiwen Huzhi Network Technologies Co. Ltd. 2017. Seafile: Source code. https://github.com/haiwen/seafile.
[42] Rob Mason. 2011. Cloud Storage Isn't Cheap: How the Price of Cloud Storage Compares to Traditional Storage. Retrieved December 10, 2018 from http://www.nasuni.com/blog/39-cloud_storage_isnt_cheap_how_the_price_of_cloud.
[43] Rob Mason. 2011. Dirty Secrets: 5 Weaknesses of Cloud Storage Gateways. Retrieved December 10, 2018 from http://www.nasuni.com/blog/28-dirty_secrets_5_weaknesses_of_cloud_storage.
[44] Mozilla Developer Network (MDN). 2017. JavaScript Tutorials, References, and Documentation. Retrieved December 10, 2018 from http://developer.mozilla.org/en-US/docs/Web/javascript.
[45] Dutch T Meyer and William J Bolosky. 2012. A study of practical deduplication. *ACM Transactions on Storage* 7, 4 (2012), 14.
[46] Microsoft Developer Network (MSDN). 2017. BITS Upload Protocol (Windows). Retrieved December 10, 2018 from https://msdn.microsoft.com/en-us/library/aa362828(v=vs.85).aspx.
[47] Athicha Muthitacharoen, Benjie Chen, and David Mazieres. 2001. A low-bandwidth network file system. In *ACM SIGOPS Operating Systems Review*, Vol. 35. ACM, 174–187.
[48] Calicrates Policroniades and Ian Pratt. 2004. Alternatives for detecting redundancy in storage systems data. In *Proceedings of USENIX ATC*. USENIX, 73–86.
[49] Varun S. Prakash, Xi Zhao, Yuanfeng Wen, and Weidong Shi. 2013. Back to the future: Using magnetic tapes in cloud based storage infrastructures. In *Proceedings of Middleware*. ACM/IFIP/USENIX, 328–347.
[50] David Rosenthal. 2012. Bandwidth costs for cloud storage. Retrieved December 10, 2018 from http://blog.dshr.org/2012/11/bandwidth-costs-for-cloud-storage.html.
[51] Raúl Sáiz-Laudó, Marc Sánchez-Artigas, and Pedro García-López. 2017. RSD: Rate-based sync deferment for personal cloud storage services. *IEEE Communications Letters* 21, 11 (2017), 2384–2387.
[52] Phlip Shilane, Mark Huang, Grant Wallace, and Windsor Hsu. 2012. WAN-optimized replication of backup datasets using stream-informed delta compression. *ACM Transactions on Storage* 8, 4 (2012), 13.
[53] Craig Smith. 2017. By the Numbers: 22 Staggering Dropbox Statistics. Retrieved December 10, 2018 from http://expandedramblings.com/index.php/dropbox-statistics.
[54] Haowen Tang, Fangming Liu, Guobin Shen, Yuchen Jin, and Chuanxiong Guo. 2015. UniDrive: Synergize multiple consumer cloud storage services. In *Proc. of Middleware*. ACM/IFIP/USENIX, 137–148.
[55] TechTarget. 2013. Bandwidth limitations are a concern with cloud backup. Retrieved December 10, 2018 from http://searchdatabackup.techtarget.com/video/Bandwidth-limitations-are-a-concern-with-cloud-backup.
[56] Andrew Tridgell and Paul Mackerras. 1996. The rsync algorithm. *The Australian National University* (1996). Technical Report TR-CS-96-05.
[57] Maria Umbsaar. 2017. The Hidden Costs of Cloud Storage. Retrieved December 10, 2018 from https://www.fotofami.com/2017/07/16/hidden-costs-cloud-storage.
[58] Michael Vrable, Stefan Savage, and Geoffrey M. Voelker. 2009. Cumulus: Filesystem backup to the cloud. *ACM Transactions on Storage* 5, 4 (2009), 14.
[59] Michael Vrable, Stefan Savage, and Geoffrey M. Voelker. 2012. Bluesky: A cloud-backed file system for the enterprise. In *Proceedings of FAST*. USENIX, 237–250.
[60] Grant Wallace, Fred Douglis, Hangwei Qian, Philip Shilane, Stephen Smaldone, Mark Chamness, and Windsor Hsu. 2012. Characteristics of backup workloads in production systems. In *Proceedings of FAST*. USENIX, 33–48.
[61] Haiyang Wang, Ryan Shea, Feng Wang, and Jiangchuan Liu. 2012. On the impact of virtualization on Dropbox-like cloud file storage/synchronization services. In *Proceedings of IWQoS*. ACM/IEEE, 1–9.
[62] Guangyuan Wu, Fangming Liu, Haowen Tang, Keke Huang, Qixia Zhang, Zhenhua Li, Ben Y. Zhao, and Hai Jin. 2016. On the performance of cloud storage applications with global measurement. In *Proceedings of IWQoS*. ACM/IEEE, 1–10.
[63] He Xiao, Zhenhua Li, Ennan Zhai, and Tianyin Xu. 2017. Practical web-based delta synchronization for cloud storage services. In *Proceedings of HotStorage*. USENIX, 1–7.
[64] He Xiao, Zhenhua Li, Ennan Zhai, Tianyin Xu, Yang Li, Yunhao Liu, Quanlu Zhang, and Yao Liu. 2018. Towards web-based delta synchronization for cloud storage services. In *Proceedings of FAST*. USENIX, 155–168.
[65] Jumie Yuventi and Roshan Mehdizadeh. 2013. A critical analysis of power usage effectiveness and its use as data center energy sustainability metrics. *Energy and Buildings* 64 (2013), 90–94.
[66] Quanlu Zhang, Shenglong Li, Zhenhua Li, Yuanjian Xing, Zhi Yang, and Yafei Dai. 2015. CHARM: A cost-efficient multi-cloud data hosting scheme with high availability. *IEEE Transactions on Cloud Computing* 3, 3 (2015), 372–386.

[67] Quanlu Zhang, Zhenhua Li, Zhi Yang, Shenglong Li, Shouyang Li, Yangze Guo, and Yafei Dai. 2017. DeltaCFS: Boosting delta sync for cloud storage services by learning from NFS. In *Proc. of ICDCS*. IEEE, 264–275.
[68] Yupu Zhang, Chris Dragga, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. 2014. ViewBox: Integrating local file systems with cloud storage services. In *Proc. of FAST*. USENIX, 119–132.

Received November 2017; revised July 2018; accepted August 2018