Data Transfer Nodes for Cloud-Storage Providers

by

Soham Sinha

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

We provide a case study of current inefficiencies in how traffic to well-known cloud-storage providers (e.g., Dropbox, Google Drive, Microsoft OneDrive) can vary significantly in throughput (e.g., a factor of 5 or more) depending on the location of the source and sink of the data. Our case study supplements previous work on resilient overlay networks (RON) and other related ideas.

These inefficiencies exist in the presence of vendor-specific points-of-presence (POP), which try to provide better network performance to the clients. In fact, the existence of special-purpose networks (e.g., national research networks, PlanetLab) and complicated peering relationships between networks, means that performance problems *might* exist in many wide-area networks (WANs).

Our main contribution is to continue the cataloging of network inefficiencies so that practitioners and experimenters are aware of them. But, we also show how simple *routing detours*, can improve throughput by factors of over 3x for client-to-cloud-storage. Routing detours are implemented by adding intermediate nodes in the routing path. These special-purpose intermediate nodes are called data transfer nodes (DTNs). We have also implemented an optimization in these DTNs in the form of cut-through routing.

Although the specific inefficiencies in this paper might be transitory (and we agree with that characterization), WAN bottlenecks due to routing, sub-optimal middlebox configuration, and congestion persist as real problems to be cataloged, discussed, and addressed through the use of detours, or data transfer nodes (DTNs), or RONs. Additionally, we provide a brief overview of the beneficial routing detours in 20 PlanetLab nodes in North America.

Preface

Some parts of all the Chapters in this thesis have been previously submitted and accepted in *21st IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems* [18]. This is collaborative work with authorship shared among myself, Di Niu, Zhi Wang and Paul Lu. I have designed the experiments, conducted them on the PlanetLab platform and deduced important observations from the experimental results. The other co-authors have provided feedbacks on the design of the experiments, improvement of the methodologies and the presentation of the paper.

To my maa, dadu and MH Sir
For being the inspirations of my life.

Perfecting oneself is as much unlearning as it is learning.

– Edsger W. Dijkstra.

Acknowledgements

I would like to thank my supervisors, Prof. Paul Lu and Prof. Di Niu, for their support and guidance all along this journey. Their insightful discussions about the research have enlightened me to think in the right direction for my research. Right from the beginning, they have shown me different paths to pursue and helped me getting over any roadblocks that I faced.

A special thanks to Trellis group members, especially Nooshin Eghbal and Hamidreza Ansari. I thoroughly enjoyed our research meetings and learned a lot out of those extended sessions.

I would also like thank my friends - Debajyoti, Arnab, Dipanjan, Ratul, Sinchan, Sanket, Sankalp, Adam, Luke, Shaiful for all the moral and technological support. I would like to thank my father, brother and extended family for their continuous support to pursue my dream. I could not have come for this degree without the encouragement of my undergraduate advisor, Prof. Manas Hira. He has been and always will be one of the inspirations to motivate myself to be in the academia. Finally, this was impossible without my mother. Thanks to her for being the source of positivity in both good and bad days.

Furthermore, thank you to Cybera (www.cybera.ca), CANARIE (www.canarie.ca), the Natural Sciences and Engineering Research Council of Canada (NSERC) (www.nserc.ca), Prof. Zhi Wang from Tsinghua University and PlanetLab (www.planet-lab.org) for their support of this research.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Non-optimal and inefficient routing on the Internet is known to exist. But with networks evolving over time (e.g., special research networks, like Canada's CANARIE[1]) and different traffic types emerging (e.g., Web vs. video streams vs. cloud-storage data), it is useful to revisit and catalog the known network issues that, in the past, have motivated ideas such as resilient overlay networks (RONs) [1], and data transfer nodes (DTNs) [5].

For example, many cloud-storage providers have multiple points-of-presence (POPs) across, say, the United States of America to improve throughput for their clients. However, we found and document how the effective throughput to popular cloud-storage providers can still be far from optimal. For example (Fig. 1.1), uploading a 100 MB binary file from a University of British Columbia (UBC) PlanetLab node to Google Drive (using Google's application programming interface (API)) takes 87 seconds (s). The same file transferred from a non-PlanetLab node at the University of Alberta (UofA) to Google Drive takes 17s. Furthermore, transferring the file from the UBC PlanetLab node to the UofA non-PlanetLab node takes 19s (this is carried by default, over the CANARIE research network). Together, by using UofA as a part of a detour from UBC to Google Drive, the 100 MB file can be transferred in 36s (= 17+19) instead of 87s.

On the one hand, transferring the data from UBC to Google Drive via UofA has extra overheads because of the involvement of an additional node. It is also

[1] http://www.canarie.ca/

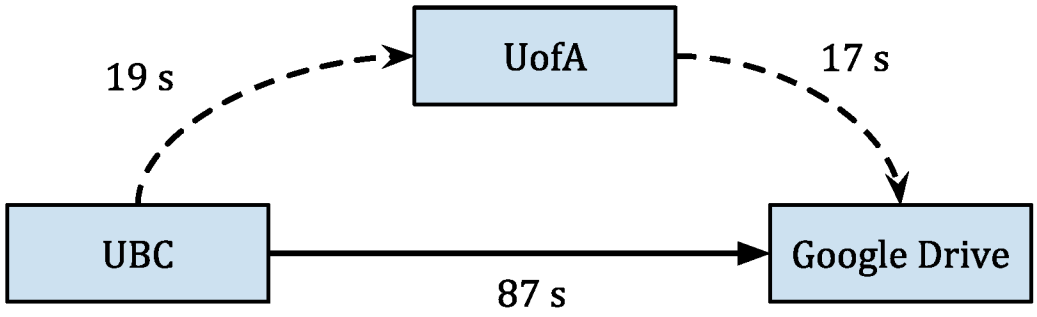


Figure 1.1: Schematic diagram of a 100 MB file transfer from UBC to Google Drive via direct route and via UofA.

counter-intuitive given the geographic backtracking involved (Fig. 3.3), and it may just be an unintended artifact of how the PlanetLab testbed is configured, combined with the presence of a high-performance research network between UBC and UofA (i.e., CANARIE). On the other hand, the performance benefits are repeatable, and we have found similar inefficiencies elsewhere in PlanetLab (Chapter 4). We certainly agree that the specific routing between PlanetLab and the world, and the complexities of network peering may have created this transitory problem. However, the ideas of overlay networks [1, 13] and the emergence of network design patterns such as Science DMZ [5] suggest that it may be necessary to explicitly identify and mitigate such inefficiencies for some workloads. The cloud-storage providers create POPs to improve performance, but explicit routing through detours and overlays may still be necessary.

Our primary contribution is to catalog and provide a case study of these performance problems. In the past several years, cloud-based storage has become popular due to the convenience, the ability to easily share files between sites and people, and the high availability of shared data. However, the speed of uploading and downloading to these cloud-storage providers vary depending on the location of the clients [6]. Therefore, no matter how transitory these performance problems might be, they can have a real impact on many users.

Also, we contribute a simple solution to the performance problem in the form of a *routing detour*, adding an intermediate data transfer node (DTN) to explicitly route the data along a faster overlay network. As a proof of concept,

our implementation of routing detours is shown to improve net transfer time performance by factors of 3 or more, depending on file size, source of the data, and cloud-storage provider. Admittedly, the scale of the improvement says more about the performance problem than it does about the importance of our contribution. However, we note that routing detours fall within a set of simple ideas that have nonetheless made it into practice. For example, Science DMZ's concept of a DTN improves performance by (in part) bypassing firewall bottlenecks, which are normally configured for non-bulk data transfers [5]. And, overlay networks [1] were conceived (in part) to explicitly control routing on an Internet with decentralized control. Our routing detours are a similar attempt to bypass bottlenecks and explicitly control routing.

In our solution of detour routing, we adopt the concept of DTNs. These DTNs open up possibilities of more optimizations. We have implemented one such optimization which is cut-through routing. Cut-through routing has been explained in Chapter 3. Optimizations like these can improve the performance further from the solution of using simple detours. As we explain in Chapter 4, cut-through routing can improve simple detour's throughput, sometimes by 50% (Fig. 4.14).

In the medium term, the identification of these inefficiencies may encourage cloud-storage providers to add additional POPs or gateways to their internal networks. Universities and institutions with the appropriate means can provide routing detours to use research networks or vendor POPs more effectively, without having to convince external parties (such as Internet Service Providers) to change their routing configurations. And, finally, our group plans to expand the functionality of our routing detours to deal with firewall bottlenecks (like Science DMZ) and to monitor and bypass dynamic bottlenecks on the WAN, as future work. We argue that careful examination of network traffic is useful for large organizations with multiple POPs. Along with meaningful and substantial knowledge of network traffic (like the information of higher share of cloud-storage traffic in a particular network), extensive experimentation of the networking infrastructure (such as done in our work) can lead to potential performance benefits.

Chapter 2

Background and Related Work

2.1 Background

In this section, we present some details of personal cloud-storage services and how we conceived of the idea of designing a special-purpose overlay network for personal cloud-storage services. We also explain the importance of such an overlay network in the context of modern network architecture such as the Science DMZ [5].

Personal cloud-storage providers provide application programming interface (API) for developers to build applications, implementing different functionalities of the cloud-storage. The cloud-storage providers also develop client applications for mobile devices, desktops, personal computers (PCs) to upload, download and synchronize local files and folders to the cloud-storage. For example, Google Drive [1], Dropbox [2], Microsoft OneDrive [3] have their own client applications for Android, iOS and desktop operating systems (OSes). The client applications for each of these services are different, based on the OS of the client-machine. Therefore, the APIs which are serviced from a cloud-storage server, are based on generalized web-based requests to serve queries from all types of clients. These web-based requests are mostly Hypertext Transfer Protocol (HTTP)-requests.

HTTP is a widely used application layer protocol for transferring different

[1] https://www.google.com/drive/download/
[2] https://www.dropbox.com/install
[3] https://onedrive.live.com/about/en-ca/download/

types of content (e.g., files, web-pages, etc.) over the Internet. HTTP has defined certain methods (e.g., `GET`, `PUT`) which are used to specify a set of actions. For example, `GET` is used to request data from the server without any other effect. Personal cloud-storage services use these types of HTTP requests to serve the files to the client applications through their set of APIs.

Most of the personal cloud-storage providers use these HTTP requests in a RESTful architecture [8]. Representational state transfer (REST) is an architectural design for web-based services. It has a set of constraints that need to be followed by the applications, to be considered as *REST*ful. For example, one constraint is supporting Stateless protocol. In Stateless protocol, the client's session is not stored on the server. Rather, whenever the client communicates with the server, it contacts the server with all the required information including the session state. However, the server can transfer a client's state internally (e.g., to a database) for authentication and other purposes. Dropbox, Google Drive and OneDrive - all of them use such RESTful HTTP request-based APIs.

For authentication purposes, major cloud-storage providers use the OAuth2 [10] authorization standard. OAuth2 simplifies and builds on the OAuth protocol. Although the specific implementations of OAuth2 may differ for different cloud-storage providers, the core idea remains the same. The server issues access tokens to the client applications on behalf of the developer (or the user of the applications). The access token is then used by the applications to authenticate and access certain services (such as cloud-storage APIs) from the cloud-storage servers.

Most of the previous research [7, 15] concerning personal cloud-storage providers, have investigated the client applications, rather than the cloud-storage APIs. Since many people use these client applications, it is worthwhile to look at these applications closely. However, when we are monitoring the client applications, in reality, we do not exactly know what we are benchmarking. In other words, it becomes a black-box testing where the black box is the client application. The client applications may have special set of permissions, features, implementations based on many factors such as the OS. For example,

previous research has shown that Dropbox provides de-duplication in its client [7]. But, we have not found any evidence of its de-duplication feature through its publicly available APIs. Therefore, those kinds of special features may only be available to the clients. We chose to benchmark the APIs because we could then customize the uploading mechanism, if needed. As we have implemented detour routing and cut-through routing, we had to use the APIs to develop our own programs for those routing mechanisms. Also, we did not want a black-box type of benchmarking mechanism, since that would not help us give any insight about how the transferring of files work. Here, a pertinent question could be why we decided to benchmark personal cloud-storage and not some other types of services.

Drago *et. al.* have shown that personal cloud-storage network traffic now-a-days accounts for as much as one-third of YouTube traffic in university-campuses [6]. On the one hand, this proportion of traffic is a significant amount of the whole network traffic. On the other hand, Dart *et. al.* have shown that in today's world, network bottleneck could be avoided if we can treat certain types of traffic specially [5]. A significant share of network traffic (such as cloud-storage network traffic) is a good candidate to be treated specially in the network, rather than generally with all other types of network traffic. Therefore, we decided that the cloud-storage services should be the type of network traffic to carry forward our experiments.

To mitigate routing inefficiencies on the Internet and improve the performance of the cloud-storage services, we revisited the ideas of overlay networking [12, 1]. Although overlay network is an old concept, it becomes important in the context of the new Science DMZ [5] network architecture. The Science DMZ design proposes deployment of special-purpose data transfer nodes (DTNs) in Local Area Networks to ensure high-speed data-transfer in Wide Area Networks. DTNs are special machines configured for high-speed data-transfer. These machines sit somewhere in between a network route. This is why we call the DTNs also as the intermediate nodes. These terms are used interchangeably through out this thesis. In case of Science DMZ, DTN is confined to the local area network. We extend the idea of DTN and make

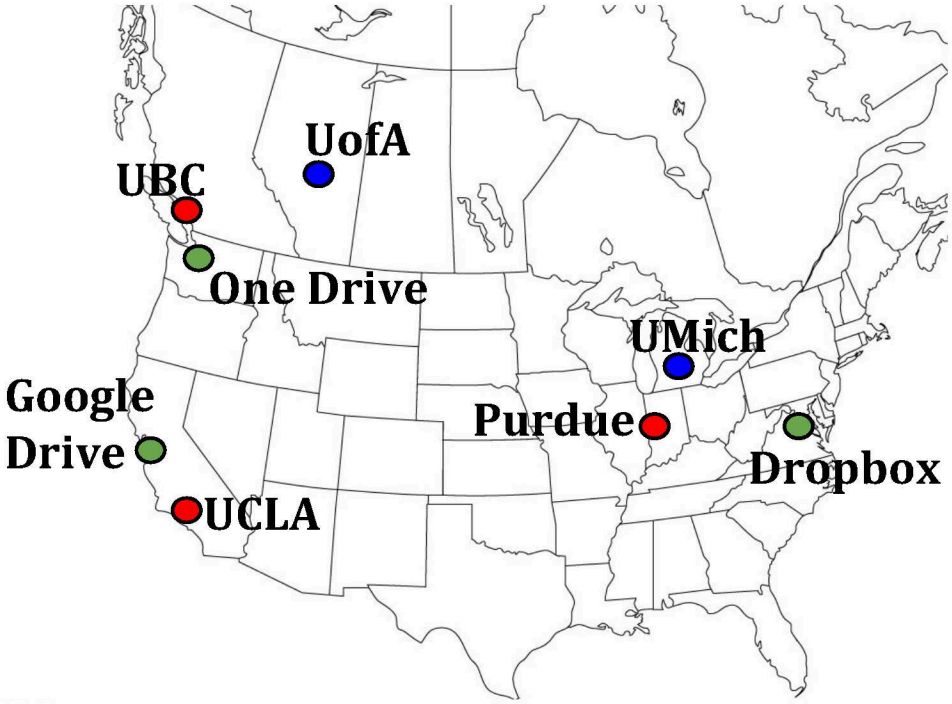


Figure 2.1: Locations of clients, intermediate nodes and cloud-storage servers. Same as Fig. 3.3

.

it a part of wide-area data-transfer. Effectively, DTNs are related to the idea of overlay networks because we are not letting the Internet to fully decide the overall route from the source to the destination.

We form an overlay of nodes in the wide area networks, where a few particular nodes can act as DTNs. In our created network of nodes, each node is a client. The clients are the source of the data to be uploaded on the cloud-storage servers such as Dropbox, Google Drive. For example, Fig. 2.1 shows multiple locations of nodes such as University of Alberta (UofA or UAlberta), University of British Columbia (UBC), University of Michigan(UMich). It also features the cloud-storage servers geographically in different places marked as Google Drive, Dropbox and OneDrive. Like the set-up in Fig. 2.1, some nodes can act as DTNs so that the data is not uploaded directly from a client to the cloud-storage server. Data will first be transferred from a client to a DTN, and then from the DTN to a cloud-storage server. For example, we might upload files from UBC to OneDrive via UofA and not directly to OneDrive server. As

we explain in Chapter 4, using the DTNs as a part of the detour routing may improve the speed of data-transfer in many cases.

There are also other benefits of using the DTNs. There can be optimizations and improvements made in the DTNs. One example of such optimizations is implementing cut-through routing [14]. We discuss the implementation of cut-through routing more in Chapter 3.

We have conducted our experiments on the PlanetLab experimental platform. PlanetLab is an well-known platform where thousands of nodes across the globe are connected. We have chosen PlanetLab because "One of PlanetLab's main purposes is to serve as a testbed for overlay networks." [4]. Our experimental set-up on PlanetLab has been described in Sec. 3.2.

2.2 Related Work

The performance of cloud-storage providers has been studied in the past from the client-side point-of-view by many researchers in the past. Most of the previous work focus on the specific details of the client applications of the cloud-storage providers and improvements concerning these applications. In this section, we provide an overview of the previous work in this area and distinguish our work from the previous research.

Li *et. al.* examined the data synchronization traffic for the client applications of six cloud-storage providers, Google Drive, OneDrive, Dropbox, Box, Ubuntu One and SugarSync [15]. The authors coined a novel metric, namely **T**raffic **U**sage **E**fficiency, to compare different cloud-storage services. They intercept and monitor the file-synchronization traffic for both desktop and mobile client applications. They make some important observations regarding the synchronization traffic such as more number of small files translates into poor *TUE*, the synchronization traffic is not optimal for several major players in personal cloud-storage services, and it is financially burdensome not only for the users but also for the cloud-storage providers. Although these observations and probable improvements are important, implementing these

[4]https://www.planet-lab.org/about

suggestions require the involvement of the cloud-storage providers. Therefore, users of the these cloud-storage providers cannot actively do much according to these observations. However, following our experimentation, large organizations with multiple locations of presence or cohort of organizations can build their own overlay network to rip out throughput benefits for cloud-storage network-traffic. Our experimental set-up is discussed in more details in Chapter 4.

Drago *et. al.* characterize network traffic to the cloud-storage providers, mainly Dropbox, by doing passive measurements from different locations in Europe [7]. The authors first establish that Dropbox is more popular than other cloud-storage providers in the examined locations. However, it is worthy to note that Google Drive and OneDrive (previously, Sky Drive) were just launched when the experiments were conducted. Since then, Google has started providing unlimited storage in Google Drive for its Google Apps Education services. Sky Drive has also improved their services in different aspects, such as user-interaction, advanced set of APIs. These factors may have been encouraged greater adoption of these services. Another observation made by Drago *et. al.* is that the share of Dropbox traffic is equivalent to one-third of YouTube traffic in university campuses [7]. As explained in the previous section, the knowledge about the Dropbox's traffic share has encouraged us to experiment with the cloud-storage network traffic.

Apart from the client-side performance measurements, there is also research on the cloud-storage services from a server-side point-of-view. For example, Wang *et. al.* investigated the cause of synchronization delay in the cloud-storage services like Dropbox [20]. They found that the virtualized environment in Amazon EC2 and the segregation of storage and computation have severe effect on the delay-issues. Furthermore, Bergen *et. al.* provide a performance analysis of Amazon AWS [3] which is used by cloud-storage providers and applications. They show in their paper how the improvements in the server-side of the cloud-storage providers can be overridden by the client bandwidth and related factors on the client-side. It is important to note that the implementation of any suggestions for the improvement on the server-side

needs the intervention of the cloud-storage providers which are usually large companies, reluctant to make quick changes or fixes on the server-side.

Our work is different from various attempts to use multiple network paths simultaneously (e.g., Begen *et. al* [2]) to improve latency, loss, or other properties. Most of these works target the streaming media contents. Therefore, they also focus on the media encoding strategies and optimizations. Additionally, these papers talk about using more than one path for delivering the contents to the clients because the single path is often not the best one. However, our presented ideas of routing detours pick a single path, known for improved performance to a particular cloud-storage provider. Future use of multiple paths would require changes to the cloud-storage provider's API significantly.

The main concept behind our proposed design of detour routing is related to the idea of overlay networking [1, 12]. Overlay network is an well-established concept where multiple nodes form an overlay on top of the existing networking infrastructure. Forming overlay is beneficial because it has existing knowledge of the network and its problems. Therefore, it can get over the problems to accomplish efficiency. Resilience Overlay Network or RON [1] is used to provide flexibility in the presence of networking problems such as link outages, lower throughput, routing anomalies, etc.

RON is generally applied on the application layer of the networking stack. The idea of overlay networking has been applied to many kinds of systems such as media streaming [19] [22], peer-to-peer network schemes [17] [4] [16]. Many of them try to ameliorate the longer latency issues.

Ideas of overlay networking now need a careful revision to be applied on different applications because of the emergence of new network designs like the Science DMZ [5]. Dart *et. al.* propose the Science DMZ network architecture which enables high-speed data-transfer. The Science DMZ network model recommends a set of network configurations, security suggestions, performance-based tools to be deployed on the local area network. The authors also show that their recommendations are useful in detecting networking problems in different networks such as universities, research labs. One of the important recommendations made in the Science DMZ paper is to deploy a special node

at the terminal of local area network. This special node is configured to transfer data in high speed, and hence called a data transfer node (DTN). We conceptualized the idea of DTNs to implement it in Wide Area Networks in our work, instead of confining it to local area network as it has been in the Science DMZ work.

In the DTNs, we have implemented one improvement in the form of cut-through routing [9]. Normally, a DTN would not start transferring a file to the destination until the whole file is received in the DTN from the source. This technique of file-transfer is called Store-and-Forward. However, in cut-through routing, the DTN starts the file-transfer even though the whole file is not present in the DTN. The DTN transfers the available part of the file to the destination. In the meantime, the source machine also transfers the rest of the file to the DTN. This technique of simultaneous data-transfers is called cut-through routing.

Another use of DTNs and overlay networking, although not in this thesis, is to address latency anomalies such as the Triangle Inequality Violation (TIV). The Triangle Inequality is the geometrical property which states that the sum of the length of two sides of a triangle is greater than the length of the other side of the triangle. Intuitively, one expects the most direct network path to have less latency than a path which covers a greater geographical distance. However, previous work has shown that TIV for latency exists on the Internet [23] [21] [13]. Zheng *et. al.* make the point that TIVs can be exploited for performance improvments [23].

However, TIV is not applicable for bandwidth. Bandwidth is not inherently a property of the distance between nodes and depends on other factors such as networking policies, infrastructure and other design decisions. In contrast, round trip time (i.e., latency) often depends largely on the distance between two nodes. In our work, we discover routing inefficiencies on the Internet, and show that we can improve the bandwidth of a particular type of network traffic (cloud-storage traffic) by choosing an alternate route.

Chapter 3

Experimental Set-up

In this chapter, we describe some details of the experimental set-up and also some of the related information. We first present some of the file-transfer APIs and describe how they work. It is important to understand the mechanisms of these APIs for different reasons such as enabling optimizations. We then show our experimental design in Sec. 3.2. Furthermore, we also delineate how cut-through routing works in our design scenario.

3.1 Cloud-storage APIs

Using the RESTful APIs, developers can access certain web-services from the cloud-storage providers. Programming languages which support the HTTP requests (e.g., Java, Python, etc.) can be used to access these APIs. For our experiments, we have used Java as our programming language and used the official libraries released by Google and Dropbox. For OneDrive, we used and modified an open-source Java library [1] as OneDrive does not have any official Java client library during the time of our experiments. For our experiments, we focus on the file-transfer operations in the API libraries, i.e., uploading a file and downloading a file from the server. We discuss the details of a few Dropbox APIs. Google Drive and OneDrive APIs follow almost similar mechanism.

The Core Dropbox API has enabled the developers to upload and download files by using the HTTP requests. The Core APIs have other functionalities as

[1] https://github.com/tjeerdnet/OneDriveAPI

Table 3.1: Dropbox File-transfer APIs in Java Client Library.

Function Name	**Description**	**Note**
`uploadFile`	Uploads a specified local file to the remote Dropbox server	If the file-size is greater than 8MB, it internally uses chunked-upload APIs to upload the file with chunk-size of 4MB.
`getFile`	Downloads a remote file from Dropbox server	No mechanism of using multiple chunks
`chunkedUploadFirst`, `chunkedUploadAppend`, `chunkedUploadFinish`	Uploads a single file in multiple chunks	An upload ID is set when the first chunk is uploaded via `chunkedUploadFirst`. Subsequent chunks are sequentially appended with `chunkedUploadAppend`, corresponding to the upload ID.

well. We will be concentrating on upload and download functions only, because these are the major file-transfer operations. We have used the Java Client API [2], officially developed by Dropbox. Table 3.1 summarizes the functions that we have used from the Java client library.

`uploadFile` and `getFile` are two straightforward functions which are used for uploading and downloading a file. These functions take the remote-server file-path and local file-path for file-transfer operations. For example, the following code snippet shows how to upload a file to Dropbox using `uploadFile`.

```
File localFile = new java.io.File("myLocalFile.txt");
FileInputStream inputStream = new FileInputStream(localFile);
String remoteFilePath = "/myRemoteFile.txt";
DbxEntry.File uploadedFile = DBclient.uploadFile(remoteFilePath,
    DbxWriteMode.force(),
    localFile.length(),
    inputStream);
```

[2]https://github.com/dropbox/dropbox-sdk-java

Additionally, Dropbox provides an alternative way to upload a file. Developers can split a file in multiple chunks and upload those separate chunks sequentially. This is achieved by using the function `chunkedUploadFirst`. The detailed mechanism of chunked-upload is explained later in Sec. 3.1.1. Important point to be noted is that the normal file-uploading function, `uploadFile` falls back to chunked-upload whenever file-size is greater than 8 MB. We discovered this behavior when we explored the source-code of `uploadFile`. In case of the fall–back option of `uploadFile`, it uses 4 MB chunks to upload the same file. In our opinion, Dropbox employs this kind of mechanism to increase the reliability of its API. If there is an interruption while uploading a big file without any chunks, the uploaded bytes up to the point of interruption would be wasted. That is why chunked-upload is a safer option. However, Dropbox does not have an option to download files in multiple chunks. This strategy of using direct upload via `uploadFile` for smaller files is specific to Dropbox, and neither to Google Drive nor OneDrive. Google and OneDrive have separate APIs for chunked and direct upload.

3.1.1 Details of Chunked-upload Mechanism

Chunking mechanism provides a safer option to upload a file in multiple chunks so that even if there is an interruption during the upload, we do not lose much of the previously uploaded bytes (hence, bandwidth). The size of the chunks is also important for throughput. Higher throughput can be obtained using larger chunks. However, in case of interruption, we will lose more with large-sized chunks.

Fig. 3.1 explains the mechanism behind chunked-upload of Dropbox. In this figure, the edges represent requests or responses, and the rectangular boxes are nodes. In case of client to server requests, labels have structures like : `function-name`, `parameters`. For example, `chunkedUploadAppend` request has 3 parameters. They are `uploadID`, `remote offset` (a number), `new chunk` (only the size of the chunk is written inside the third-brackets in the figure).

Chunked-upload starts with a function call to `chunkedUploadFirst`. This function takes the first chunk as parameter and returns an upload identifier

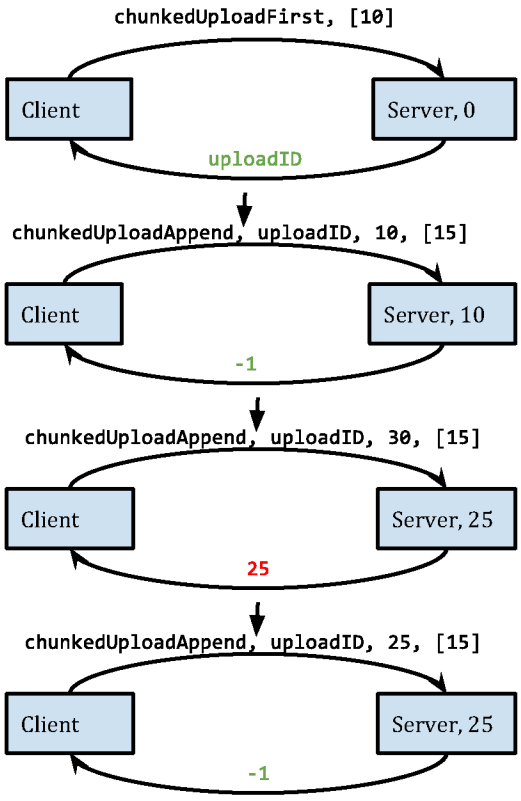


Figure 3.1: Chunked-upload mechanism of file upload in Dropbox.

(ID) to the client. In the example in Figure 3.1, the size of the first chunk is 10 bytes (size is mentioned inside third-brackets in the label of the request). Upon receiving a new upload request, the Dropbox server creates an upload ID and returns it to the client. The server also keeps track of the size of uploaded bytes corresponding to an upload ID. In Fig. 3.1, this size is initially set to 0 (separated by a comma in the "Server" rectangular box).

Next, when the client wants to append a chunk to the uploaded bytes on the server, it has to provide the correct server offset from which appending should occur. To append a chunk, the client invokes `chunkedUploadAppend` function with the recently created upload ID, the remote (server) offset and the new chunk as parameters. In our example, the second sub-figure from top in Figure 3.1 shows that `chunkedUploadAppend` has 10 as the remote offset parameter (after the `uploadID` parameter). The size of new chunk is 15 bytes (next parameter, inside third-brackets).

After that, Dropbox server matches the remote offset parameter received

from the request of `chunkedUploadAppend` with its own saved offset. If both of them matches, the chunk is then appended to the server, and the server offset is increased by the size of the current chunk. In the example in second sub-figure, the parameter offset (10) and the actual remote offset (which is also set to 10 inside "Server" box, previously by `chunkedUploadFirst`) matches. Therefore, the server successfully appends the chunk. It sends -1 to client as an acknowledgement of success. The server offset is then increased by 15 (size of current chunk), which can be seen in "Server" box of the third sub-figure.

In the third sub-figure of Fig. 3.1, the client sends a wrong remote offset (30) as its parameter. This does not match with 25 which is the actual remote offset in the server, set by the previous step. In case of wrong and unmatched offset such as this one, the Dropbox server discards the chunk and returns the actual offset from the server (which is 25 in our case). Finally, in the last sub-figure the client sends the correct offset 25 to the server. Then, the server accepts the chunk and returns -1 as usual.

The client finally has to call `chunkedUploadFinish` with the upload ID to complete the uploading process. Then the server commits the file into itself, and we can access the same file in our Dropbox folder. This final `chunkedUploadFinish` call has not been shown in the figure.

3.1.2 Discussion and Examples

Traditionally, users utilize the official client applications of different cloud-storage providers to upload, download their files to or from the cloud-storage servers from or to their local machines. The client applications are available for different OSes. For example, Dropbox has their client applications for desktop OSes [3] (Windows, Linux, Mac) and mobile OSes [4] (iOS, Android, Windows Mobile). It also supports web-browsers in both mobile and desktop. On the one hand, these client applications may internally use the cloud-storage APIs to connect to the cloud-storage servers. On the other hand, application developers directly use the cloud-storage APIs to connect to the cloud-storage

[3] https://www.dropbox.com/install
[4] https://www.dropbox.com/mobile

servers for their own applications. Therefore, these APIs are the underlying layer of connections between the clients and the cloud-storage servers, most possibly in both the cases. Thus, we selected these APIs to measure the performance of personal cloud-storage services.

We have already given an example of code-snippet for uploading file to Dropbox. Following is a similar example for Google Drive:

```
java.io.File localFile = new java.io.File("myLocalFile.txt");
String remoteFilePath = "myRemoteFile.txt";

com.google.api.services.drive.model.File fileMetadata =
    new com.google.api.services.drive.model.File();
fileMetadata.setTitle(remoteFilePath);

InputStreamContent mediaContent = new InputStreamContent(null,
    new BufferedInputStream(
    new FileInputStream(localFile)));
Drive.Files.Insert insert = driveService.files().
    insert(fileMetadata, mediaContent);

MediaHttpUploader uploader = insert.getMediaHttpUploader();
uploader.setDirectUploadEnabled(true);

com.google.api.services.drive.model.File googleModelFile =
    insert.execute();
```

Next, we provide an example for OneDrive. As it has been implemented and customized from a third-party code-base, we have not developed all the features for this library. For example, in the following case, the remote file name is not given and assumed to be the same as the local file name.

```
java.io.File localFile = new java.io.File("myLocalFile.txt");
net.tjeerd.onedrive.json.folder.FileResponse oneDriveFile =
```

```
oneDriveClient.uploadFile(localFile, "");
//second parameter is the folder ID
//where empty means home folder
```

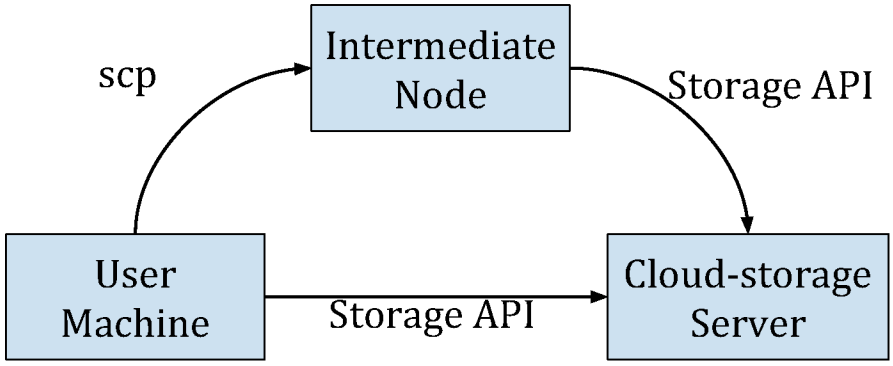
3.2 Experimental Design

Now, we explain our idea on how we propose to transfer files faster for cloud-storage services. The idea has been presented schematically in Fig. 3.2. In the figure, the *User Machine* is the client machine or the source machine from (to) where we need to upload (download) files to (from) the cloud-storage servers. Cloud-storage APIs can directly be used for the file-transfers as described earlier and shown in the figure by the horizontal line between *User Machine* and *Cloud-Storage Server*. However, we have observed that uploading directly to cloud-storage servers are sometimes slow. This can happen due to various reasons, such as low bandwidth, routing inefficiencies and other unknown bottlenecks. As we explain in our experimental results, these types of anomalies can cause significant performance degradation. In those cases, an *Intermediate Node* becomes crucial. An *Intermediate Node* can help a *User Machine* in different ways, for example, by caching or by its higher data-transfer speed. We have observed in our experiments that if we upload a file via certain intermediate node(s), then we can achieve higher throughput (lower time to transfer files) than uploading via a direct route to the cloud-storage servers. To upload files via the detour routes, we use `scp`[5] to transfer files between a *User Machine* and an *Intermediate Node* and lastly use cloud-storage APIs to upload files finally to the cloud-storage servers from the *Intermediate Nodes*. The detour has been shown in Figure 3.2 by the curved lines from the *User Machine* node towards the *Cloud-Storage Server* node via the *Intermediate Node*. *Intermediate Node* can also be called Data Transfer Node (DTN) since its main motive is to transfer files or data.

It should be noted that the files on the *Intermediate Node(s)* are always deleted before benchmarking, so there is no benefit gained from the file-transfer

[5]http://linux.die.net/man/1/scp

Figure 3.2: Schematic diagram of our experimental design for uploading data to the cloud-storage servers.



tool (such as `rsync`)'s ability to send only file deltas. And, since the file contains random data, it is resistant to any compression-based performance artifacts. Lastly, although we currently use `scp`, it can be replaced with a different file-transfer tool (e.g., `rsync`) without changing the basic ideas of our design.

We conducted our experiments on different PlanetLab nodes in North America and on our own non-PlanetLab cluster at the University of Alberta (UofA). Originally, we chose to use PlanetLab to get access to nodes in many geographical locations, and not necessarily to benchmark the performance of transferring from PlanetLab to cloud-storage providers. We realize that our inefficiencies may be very specific to how PlanetLab traffic is routed to the rest of the world, at certain sites. Given different testbeds and different peering relationships between networks, our inefficiencies may or may not even exist. However, PlanetLab *is* a widely used testbed and not just a minor entity. Therefore, we continued with our work.

Our primary metric of performance is the file-transfer time, which is directly related to throughput, of course. To compare throughput, we create differently sized binary files (using `dd` Unix utility with `random` data source) and measure the time taken to upload those files. The file sizes are 10, 20, 30, 40, 50, 60 and 100 MB. For each of the measurements, we take the mean of the last five runs among a total of seven runs. In a single run, we vary

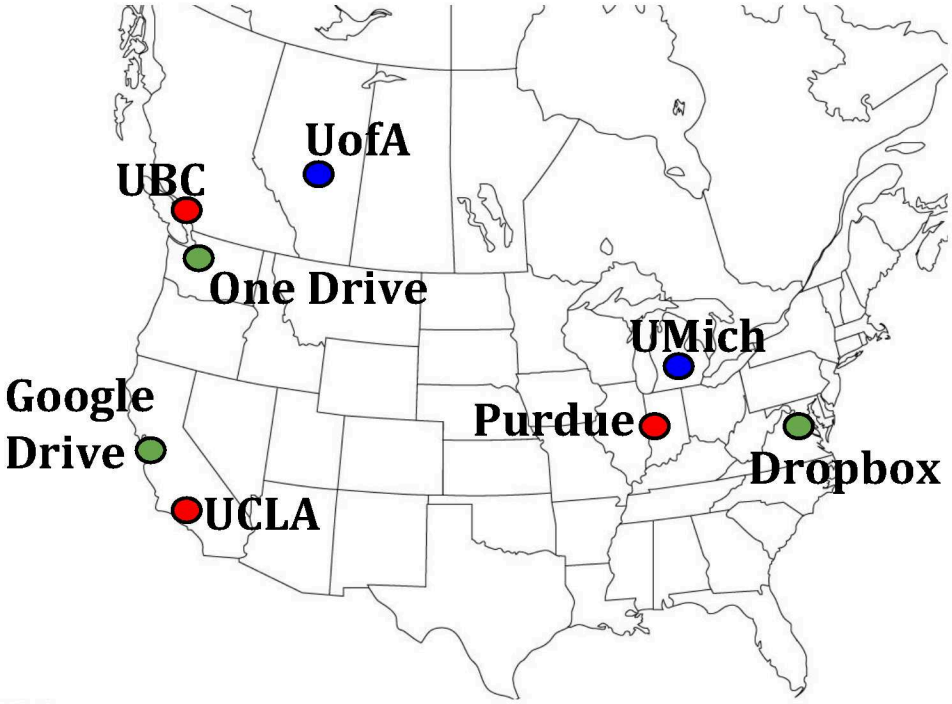


Figure 3.3: Locations of clients, intermediate nodes and cloud-storage servers. Same as Fig. 2.1.

the file size. One standard deviation has been shown as the error-bar in the figures. We have written basic file-transfer programs in Java, using the APIs of the cloud-storage providers to upload/download files. The experiments were conducted between the span of October 2015 to March 2016.

We have chosen the PlanetLab nodes based on their geographical positions. As seen from the previous research [6], the geographical location of client does have an effect on the performance for different cloud storage. However, the magnitude of the effect is not clear in the previous research. Our experiments clearly shows the effect of geographical locations for these cloud-storage services. We have chosen one PlanetLab node in Canada at the University of British Columbia, Vancouver (UBC, west coast of North America (NA)). The rest of the PlanetLab nodes are in the United States of America: University of Michigan (UMich, eastern half of NA), Purdue University (eastern half of NA), University of California, LA (UCLA, west coast of NA). We have also identified the locations of data-center servers for Dropbox, Google Drive and Microsoft OneDrive for all of our experiments. They are located at Ashburn, VA

(Dropbox), Mountain View, CA (Google Drive) and Seattle, WA (OneDrive). Fig. 3.3 shows the geographical locations (obtained from `traceroute` and IP Location Finder [11]) of all the tested clients, intermediate nodes, and cloud storage servers in our experiments.

3.3 Cut-through Routing

Cut-through routing is a known and straightforward concept in the area of data-transfer and network routing. It is an effective way to speed up the data-transfer by harnessing the pipelining and parallelism aspects in networking. We describe the cut-through routing below. We also explain how we use cut-through routing in our design scenario to increase the overall throughput.

In our experiments, we have seen that sometime it is beneficial to transfer the data via an intermediate node or DTN instead of directly uploading to the cloud-storage servers from the client nodes. To transfer a file via intermediate node, we first move the whole file from the client machine to the intermediate node. After the whole file is completely transferred to the intermediate node, we start uploading the file finally to the cloud-storage servers. This is called Store-And-Forward Routing. It is schematically presented in the left side of Fig. 3.4. In this process, we need to wait for the whole file to become available on the intermediate node or DTN before we can transfer it to the cloud-storage server. However, a part of the file may already be available on the intermediate node, even if the file is not completely transferred from the client machine. Cut-through routing takes advantage of the availability of the partial file on the intermediate node.

In cut-through routing (presented schematically in the right side of Fig. 3.4), we start transferring a file from the DTN to the cloud-storage servers, even if the full file is not available. We keep uploading the available part of the file from the DTN while the unavailable part of the file is being transferred from the client node. We use *scp* to transfer data between the intermediate node and the client machine. When *scp* transfers a file from source to destination, it creates the final file in the destination machine and keeps appending more

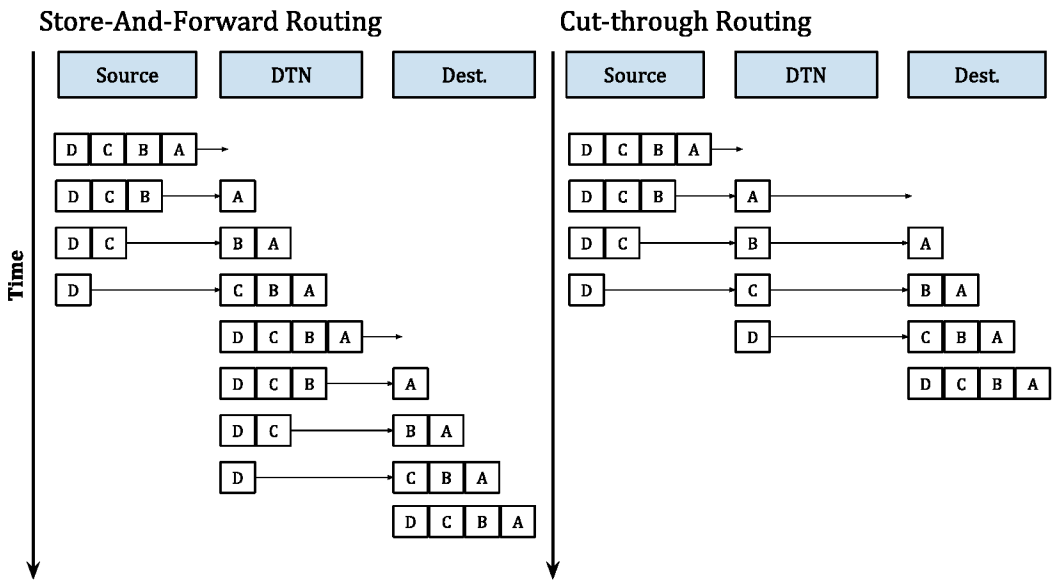


Figure 3.4: Data Transfer Technique in Store-And-Forward Routing and Cut-through Routing with respect to the data transfer node (DTN).

content from the source machine to that final file in the destination. As soon as we get some content of the final file in the intermediate node, we start uploading that content to the cloud-storage servers using the file-transfer APIs of different cloud-storage providers.

To transfer partially available files from the intermediate nodes to the cloud-storage server, we had to modify some codes for the cloud-storage APIs. Dropbox, Google Drive and OneDrive have different sets of APIs and slightly different approaches to upload files. For Dropbox, it is very simple to upload partial content to their servers. It has the support to append files for a given file ID. We use Dropbox's appending API to transfer contents continuously from the intermediate node, as the contents are delivered from the client machine. OneDrive also supports similar mechanism to Dropbox, but against a file upload URL, instead of a file ID. Google Drive does not support simple appending like Dropbox and OneDrive do. However, they have advanced set of APIs to do so, but with some constraints. For example, one of the constraints in case of Google Drive is that the size of a partial data transfer should be at least of the size of 256 KB. Therefore, if the data-transfer speed is slow between the client machine and the intermediate node, we may not have ac-

cumulated enough data to be transferred to the Google Drive server from the intermediate node. For these reasons, we modified Google Drive's open source API, to continuously monitor the transferred file in the intermediate node, for reasons such as the size of the available data crosses the threshold of 256 KB and then, we will be able to upload a part of the file to the Google Drive server.

We have implemented cut-through routing and documented the performance of it in every examined client node and intermediate node locations. As desired and explained in Chapter 4, cut-through routing increases the throughput in almost every case.

3.4 Collection of PlanetLab Statistics

While we were conducting our experiments in the PlanetLab nodes, we discovered some interesting results in this platform. We describe and explain those results in Chapter 4. Briefly, we observed that detours can improve the performance from client-to-cloud-storage servers significantly for some client nodes and some cloud-storage providers (detours alone and with other techniques such as cut-through routing). This encouraged us to investigate more such scenarios. Therefore, we chose 20 active nodes in PlanetLab to conduct brief experiments. As PlanetLab is a fairly old and established experimental platform, many of its nodes are often not maintained and become inactive. We chose only the first 20 active nodes that we found in North America and have no location bias in the selection process. In those 20 nodes, we measured the transfer-times for a 10 MB binary file for all those 20 nodes, using `scp`. We also measured the uploading time to all three cloud-storage providers (Dropbox, Google Drive, OneDrive) from those 20 PlanetLab nodes, using the cloud-storage APIs. This helps us to recognize beneficial detours among the 20 PlanetLab nodes, if there are any. Detailed results of these measurements are presented in next chapter in Sec. 4.6.

Chapter 4

Results and Analysis

In this chapter, we present our results from the experiments conducted on PlanetLab nodes spread across multiple locations in North America. We explain our findings regarding network speed to three popular personal cloud-storage services including Dropbox, Google Drive and Microsoft OneDrive. Our experimental results demonstrate that the common intuition that network speed to cloud-storage providers largely depends on the proximity is not always true—sometimes a detour can save transfer time, although the relative benefits of detour routing may vary for different clients, services, and even file sizes. We present our results for three case studies of uploading files from (A) University of British Columbia (UBC), (B) Purdue University, and (C) University of California, Los Angeles (UCLA) to the mentioned storage services, respectively, and summarize the relative performance of various routes for different file sizes in Table 4.1. We also present Table 4.5 which captures the experimental best routes for all of the three client-locations and cloud-storage services in geographical maps.

While detours may not always improve the throughput, they open up the possibility of using an intermediate node between the client machine and cloud-storage server. In the intermediate node, we have added an optimization called cut-through routing which is already explained in the previous chapter. Briefly, cut-through routing enables simultaneous data-transfers to and from the intermediate node. The results of enabling cut-through routing has also been discussed later in Sec. 4.5.

Table 4.1: Summary of the average file transfer times from three client locations to the three cloud-storage providers, using different routes. See also Table 4.5.

Services / Clients	**Google Drive**	**Dropbox**	**OneDrive**
(A) UBC	Fastest: via UofA Fast: Direct Slowest: via UMich	Fastest: Direct Fast: via UofA Slowest: via UMich	Fastest: Direct Fast: UofA Slowest: via UMich
(B) Purdue	Fastest: via UofA and via UMich Slowest: Direct	Fastest: Direct Slowest: via UofA and via UMich [1]	Fastest: Direct Slowest: via UofA and via UMich [2]
(C) UCLA	Fastest: Direct Fast: via UofA Slowest: via UMich [3]	Fastest: Direct Fast: via UofA Slowest: via UMich	Fastest: Direct Fast: via UofA Slowest: via UMich [4]

[1] Exceptions: 40 MB - Slowest: via UMich, Fast: Direct, Fastest: via UofA, 60 MB - Slowest: Direct, Fast: via UMich, Fastest: via UofA

[2] Exceptions: 10, 60 MB - Slowest: Direct and via UMich, Fastest: via UofA, 100 MB - Slowest: Direct, Fast: via UofA, Fastest: via UMich

[3] Exceptions: 10, 20 MB - Slowest: via UofA, Fast: Direct, Fastest: via UMich, 100 MB - Slowest: via UofA, Fast: via UMich, Fastest: Direct

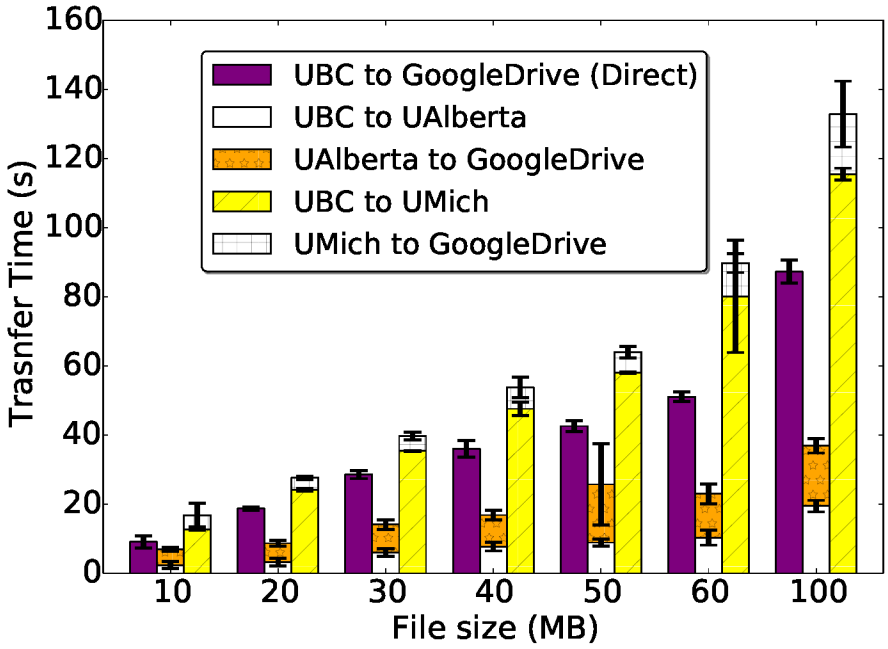
[4] Exceptions: 10 MB - Slowest: via UofA, Fast: via UMich, Fastest: Direct

4.1 UBC: Direct Uploads vs. Detours

First, we present our results measured from the PlanetLab node at the UBC. We upload our test files from UBC to Dropbox, Google Drive, and OneDrive, respectively. Our baseline is to use the APIs of these services to upload files directly from UBC to their storage servers. We compare such direct uploads with detoured transfers which route traffic via another intermediate node, as has been described in Chapter 3. For this experiment, our candidate intermediate nodes include our computing cluster (non-PlanetLab) at the University of Alberta (UofA or UAlberta) and a PlanetLab node at the University of Michigan (UMich). We only consider one extra hop in our experiments.

Fig. 4.1 compares the performance of a direct upload from UBC to Google Drive, versus detoured uploads via UofA and UMich. For all file sizes, the

Figure 4.1: Upload performance from **UBC to Google Drive** (direct routes and detours) [Error bar: One Standard deviation].



direct upload from UBC is slower than the two-hop indirect route via UofA in Edmonton, Alberta, Canada. For example, 100 MB file is uploaded in 87s via direct route, and the same 100 MB file is uploaded in (19s + 16s) = 35s via UofA-detour. This is counterintuitive in the sense that one might expect direct uploads to be faster than indirect routes. However, more intuitively, a direct upload from UBC to Google Drive is faster than routing via UMich. The 100 MB file takes 115s to be transferred from UBC to UMich, and 17s to be delivered from UMich to Google Drive server. The total time for UMich-detour is 132s. Although in general network speed is very fast and not the bottleneck from UMich to Google Drive in these set of experiments, uploads from UBC to UMich are too slow. Thus there is no benefit of using UMich as an intermediate point from UBC to Google Drive. In contrast, UofA is a good intermediate node, since the bandwidth is decent both from UBC to UofA and from UofA to Google Drive. Table 4.2 shows the average transfer times from UBC to Google Drive for direct and detour routes. It also includes the relative gain/loss in transfer times for the detoured routes inside the parenthesis, compared to direct routes. In that table, we can look at the data for 100 MB file. Detour

Table 4.2: UBC-to-Google Drive Average Transfer Times through different routes.

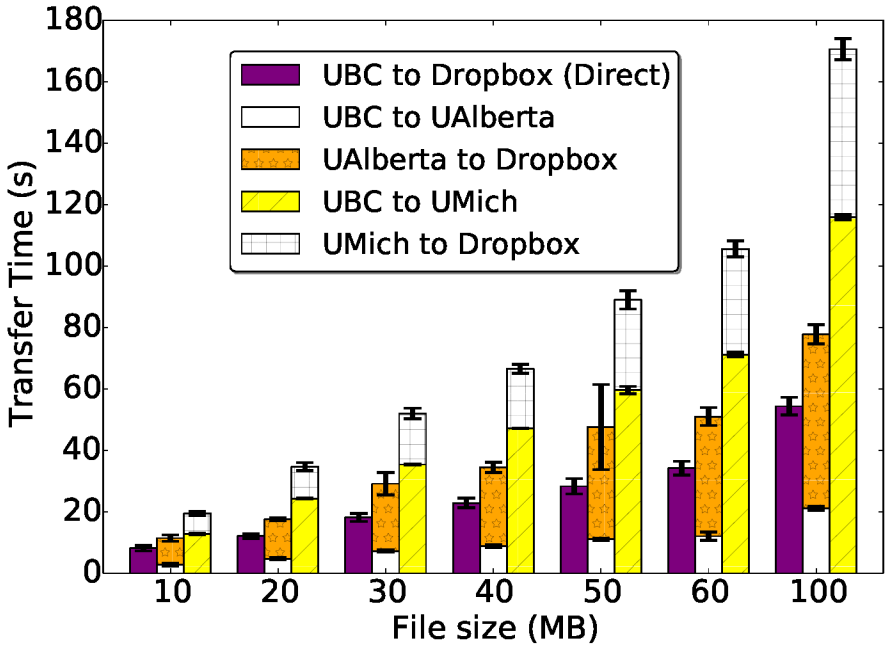
File size (MB)	**Direct (s)**	**via UofA (s) [%]**	**via UMich (s) [%]**
10	9.5	6.5 [-31.5%]	15.4 [+62.9%]
20	18.6	8.2 [-55.5%]	27.7 [+48.9%]
30	28.7	13.8 [-51.7%]	39.1 [+36.6%]
40	36.9	17.4 [-52.8%]	51.9 [+40.7%]
50	42.3	19.4 [-54.1%]	63.7 [+50.7%]
60	51.1	22.0 [-57.0%]	80.7 [+57.9%]
100	86.9	35.8 [-58.8%]	132.2 [+52.1%]

via UofA reduces the file-transfer time by 58.8% which is given inside the parenthesis in third column. But detour through UMich increases the transfer-time by 52.1% for 100 MB file. Similar cases can be observed for other file sizes. Therefore, we can conclude that UofA-detour is able to improve the transfer times by more than 50% in almost all of the cases, while UMich-detour is not able to.

In order to understand UofA-detour's enhanced performance over direct route, we first look at the geographical locations of UBC, UofA and Google Server. Through `traceroute` to Google Drive from UBC (Fig. 4.3) and from UofA (Fig. 4.4), we can see that all three direct and indirect upload modes lead to the same Google Drive server located in Mountain View, CA, USA, based on IP Geo-location services [11]. From Fig. 3.3, we can see that going from UBC to Google Drive in Mountain View via UofA is a significant *geographical* detour. Yet, the detour has a higher bandwidth than the direct route. This observation of a detour's possibility of having more bandwidth implies that geographical proximity or a direct route may not necessarily indicate higher upload bandwidth than detoured transfers. Such an observation also brings about the opportunities to increase network speed by judiciously choosing an intermediate routing node in an overlay network for cloud storage services.

Moreover, even if the routes to be compared are both direct routes, Fig. 4.1 reveals that geographic proximity may not be positively correlated with the network speed. For example, although UBC is located closer to the same Google Drive server in Mountain View than UofA is, UBC has a slower upload

Figure 4.2: Upload performance from **UBC to Dropbox** (direct routes and detours) [Error bar: One Standard deviation].



speed to Google Drive. It is worth noting that the outgoing bandwidth at UBC is not really the bottleneck here, which can be observed from the file transfer time between UBC and UofA in Fig. 4.1. For 60 and 100 MB files, it takes only 9.3s and 19s to transfer the files from UBC to UofA.

Figure 4.3: UBC to Google Drive Server traceroute.

```
traceroute to www.googleapis.com (216.58.216.138)
 1  142.103.2.253 (142.103.2.253)
 2  a0-a1.net.ubc.ca (142.103.78.250)
 3  anguborder-a0.net.ubc.ca (137.82.123.137)
 4  345-IX-cr1-UBCAb.vncv1.BC.net (134.87.0.58)
 5  vncv1rtr2.canarie.ca (199.212.24.64)
 6  google-1-lo-std-707.sttlwa.pacificwave.net (207.231.242.20)
 7  209.85.249.32 (209.85.249.32)
 8  216.239.51.159 (216.239.51.159)
 9  sea15s01-in-f138.1e100.net (216.58.216.138)
```

Continuing the analysis based on `traceroute` (Figures 4.3 and 4.4), both UBC-to-Google Drive and UofA-to-Google Drive network routes cross the middlebox `vncv1rtr2.canarie.ca (199.212.24.1)` once. However, the traffic from UBC is routed through the *pacificwave*[1] (6th hop in Fig. 4.3) network

[1]http://pacificwave.net/

Figure 4.4: UofA to Google Drive Server traceroute.

```
traceroute to www.googleapis.com (216.58.216.138)
 1  ww-fw.cs.ualberta.ca (129.128.184.254)
 2  * * *
 3  172.26.244.22 (172.26.244.22)
 4  172.26.244.17 (172.26.244.17)
 5  core1-sc.backbone.ualberta.ca (129.128.0.10)
 6  gsb-asr-core1.backbone.ualberta.ca (129.128.0.21)
 7  uofa-p-1-edm.cybera.ca (199.116.233.66)
 8  edmn1rtr2.canarie.ca (199.212.24.68)
 9  vncv1rtr2.canarie.ca (199.212.24.1)
10  * * *
11  209.85.249.32 (209.85.249.32)
12  216.239.51.159 (216.239.51.159)
13  sea15s01-in-f138.1e100.net (216.58.216.138)
```

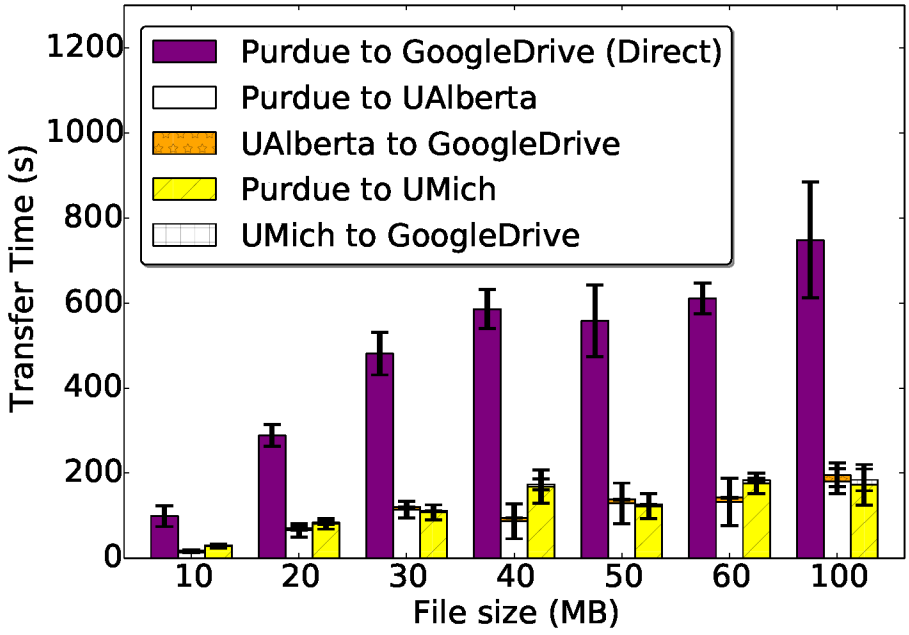
link to the Google Drive server from that middlebox, whereas the traffic from UofA is directly transferred to the Google Drive server through an unknown hop (denoted by *** in 10th hop).

However, intermediate node(s) must be carefully chosen to increase the transfer performance to cloud-storage providers. The performance gain observed for one service may not appear in another service. Fig. 4.2 plots the performance of file uploads from UBC to another cloud storage service, namely Dropbox. We can see here that direct upload outperforms both indirect routes via UofA and UMich. Thus, in this case, choosing UofA or UMich as an intermediate node could be a worse choice and is unlikely to yield performance gain. Similar results are observed for OneDrive, and hence, are not plotted here for brevity.

4.2 Purdue: Choice of Detoured Node

We now discuss the experimental results from Purdue University, with the same candidate intermediate nodes UMich and UofA, for all three cloud storage providers tested before. Tests in Purdue reveal more on how the choice of the intermediate node in detour routing is important. Purdue is located in the eastern part of North America, which is geographically far away from our previous point of experiment, UBC. However, similar to UBC for Google Drive, a detour for Purdue results in faster uploads (Fig. 4.5). And, both the detours (i.e., through either UofA and UMich) are faster than a direct upload, which is

Figure 4.5: Upload performance from Purdue to Google Drive (direct routes and detours) [Error bar: One Standard deviation].



different from the UBC case for Google Drive. And, there is no performance-based reason to prefer a detour through UofA to that through UMich or vice versa, since their transfer times are comparable. Table 4.3 shows the average transfer times and the relative gain of both the detours (with respect to the direct route) in parenthesis. UofA- and UMich-detour display performance gain in the range of 73.8% to 83.9% and 69% to 77.3%, respectively.

Choosing the best intermediate node for a detour is a multi-dimensional problem. The optimal detour depends on the location of the client/user, the available detour nodes, the cloud-service provider, and possibly even the size of the file (discussed below). For example, as discussed above, UBC-to-Google Drive via UMich is not beneficial compared to direct upload, but Purdue-to-Google Drive via UMich is better than the direct upload. Therefore, the location of the client is one of the parameters for the choice of detour node. At this time, our case study only identifies the best detour, but we have not implemented an automatic detour selection algorithm. We now point out how the size of a file can have an effect on the transfer times.

The upload behavior becomes more complicated when we analyze file up-

Table 4.3: Purdue-to-Google Drive Average Transfer Times through different routes

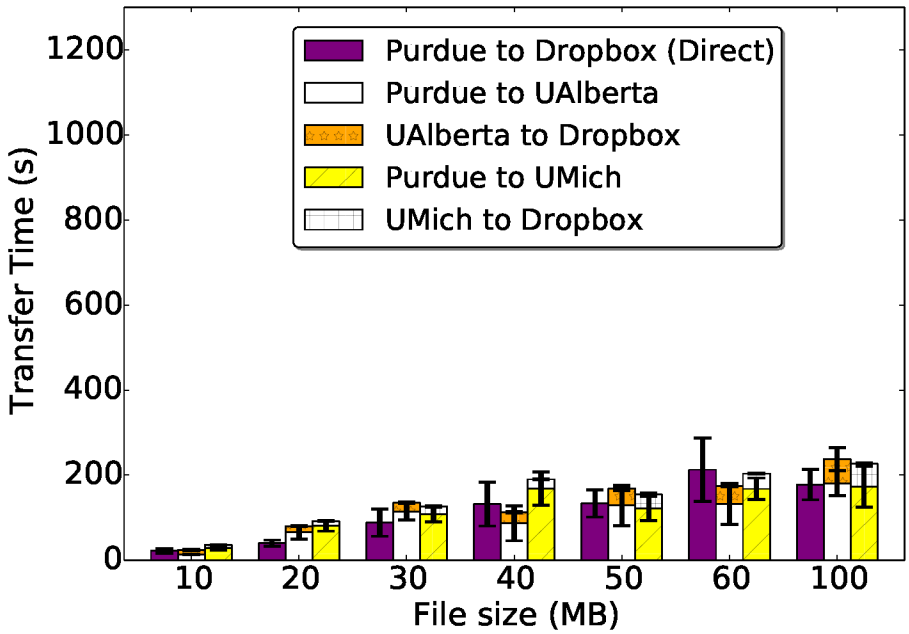
File size (MB)	**Direct (s)**	**via UofA (s) [%]**	**via UMich (s) [%]**
10	98.9	17.6 [-82.2%]	30.6 [-69.0%]
20	288.2	70.5 [-75.5%]	83.6 [-71.0%]
30	480.9	120.7 [-74.9%]	111.4 [-76.9%]
40	585.5	94.4 [-83.9%]	173.5 [-70.4%]
50	557.9	138.0 [-75.3%]	126.8 [-77.3%]
60	610.9	142.1 [-76.8%]	183.8 [-69.9%]
100	748.0	195.9 [-73.8]	184.1 [-75.4%]

loads from Purdue to Dropbox and OneDrive, as shown in Fig. 4.6 and Fig. 4.7, respectively. The relative performance of direct uploads vs. detoured uploads here *is dependent on file sizes*. In Fig. 4.6, except for 40 MB and 60 MB files, detoured uploads (2nd and 3rd bar) have higher upload time than direct upload (1st bar). Therefore, detoured transfers via intermediate nodes are generally no better than direct uploads, except for those two file-sizes. For 40 MB files, detoured uploads via UofA (112.2s) outperform direct uploads (131.6s), although a detour via UMich (190.1s) does not. For 60 MB files, both detoured uploads via UofA (174.5s) and UMich (203.8s) outperform direct uploads (212.7s).

Similarly, for file uploads from Purdue to OneDrive, as shown in Fig. 4.7, the relative gain from detoured transfers also varies as file sizes change. More evidently, for the case of OneDrive, detoured transfers via intermediate nodes can bring more benefits for larger files. For, 60 MB and 100 MB files, the detoured uploads (2nd and 3rd bar in Fig. 4.7) have lower transfer time, but for 10–50 MB files, they have higher transfer time. As a result, in such scenarios where the relative performance of different routes also depends on the size of the file to be transferred, it is tricky to decide between the direct route and detours, and choose the best intermediate node for a high speed detoured transfer.

Furthermore, we have not only evaluated the relative performance of different routes by the mean upload time taken, but have also recorded the variations of the time measurements on each route. For example, in Fig. 4.6, with 40

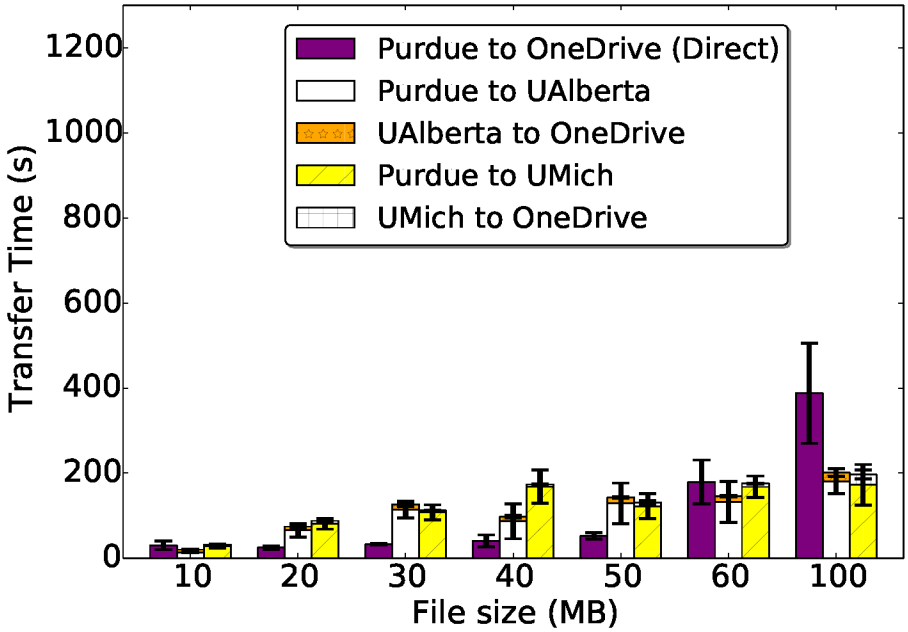
Figure 4.6: Upload performance from Purdue to Dropbox (direct routes and detours) [Error bar: One Standard deviation].



MB files, the error bar (representing one standard deviation) is quite large for direct uploads, with its lower end overlapping with the time for a detour via UofA, making it harder to decide whether direct uploads or detoured uploads are faster. Generally, a route (whether being direct or detoured) with both a lower mean transfer time and a lower transfer time variance is more desirable.

To understand the statistical significance of the measured performance and possible overlapping of detoured and direct transfer times, Table 4.4 summarizes the average transfer times from Purdue along with their standard deviations for Dropbox and OneDrive, respectively, for 100 MB and 60 MB files. If we consider the mean timings for both the direct and detour routes and add/subtract 1 standard deviation to/from it, we can see that their higher and lower end of readings overlap. We clarify this by one example. We examine the measurements for uploading the 100 MB file for Dropbox. Adding +1 standard deviation to the mean value of the upload times for Dropbox (Direct) gives (177.9 + 36.0) = 213.9s, which marks the higher end of the error bar of upload time. However, subtracting 1 standard deviation from the mean value of the detoured upload times for Dropbox gives (237.8 - 56.1) =

Figure 4.7: Upload performance from Purdue to OneDrive (direct routes and detours) [Error bar: One Standard deviation].



181.7s for UofA-detour, and (226.4 - 50.5) = 175.9s for UMich-detour. Both 181.7s and 175.9s for detours represent the lower end of their respective error bars for the upload times measurements. Although the mean upload time for direct route (177.9s) is smaller than the mean upload times for both the detoured routes (237.8s, 226.4s), the higher end of direct routes (213.9s) becomes greater than the lower ends of the detoured routes (181.7s, 175.9s). Hence, it clearly exemplifies an overlap between the direct and detoured transfer times. It is also evident in Fig. 4.6. Similar observations can be made for other cases in Table 4.4. Because of this significant overlap, we may not choose to rely on any detours in these types of scenarios. Rather, the direct routes can be better than the other choices in these cases due to the unsure benefits of the detours.

4.3 UCLA: Where Detours Do Not Help

It is worth noting that in all the above experiments, the performance benefit is not due to the network configurations on specific PlanetLab nodes, as the

Table 4.4: The mean and standard deviation of upload times (in seconds) from Purdue. Same data as in Figures 4.6 and 4.7, but quantitatively for 60 MB and 100 MB only.

File-size (MB)	**Type**	**Mean (s)**	**Standard deviation**
100	Dropbox (Direct)	177.9	36.0
	Dropbox (via UofA)	237.8	56.1
	Dropbox (via UMich)	226.4	50.5
	OneDrive (Direct)	387.7	117.8
	OneDrive (via UofA)	201.9	38.6
	OneDrive (via UMich)	197.2	58.2
60	Dropbox (Direct)	212.7	74.9
	Dropbox (via UofA)	174.5	50.2
	Dropbox (via UMich)	203.8	26.9
	OneDrive (Direct)	179.4	51.5
	OneDrive (via UofA)	145.9	50.1
	OneDrive (via UMich)	175.4	26.1

outgoing bandwidth of the above tested PlanetLab nodes did not form the bottleneck in the corresponding experiments. However, we need to emphasize that it is harder for the detoured transfer to improve the throughput where the network bottleneck is the last mile bandwidth of the end host being tested. The experimental results of uploading files from UCLA to Google Drive and Dropbox have illustrated this phenomenon. Our measurement suggests that the PlanetLab node at UCLA may have limited bandwidth. Fig. 4.8 shows the performance of uploading files from UCLA to Google Drive. In that figure, the bottom bars representing the file transfers from UCLA to all other locations including the Google Drive server, UofA, etc., take a long time. In this case, there is little chance to reduce transfer time by identifying faster detours, since the network bottleneck is (we speculate) UCLA's outgoing bandwidth from that PlanetLab node. Similar observations are made for Dropbox and OneDrive. We plot the results for Dropbox in Fig. 4.9 and skipped the results of OneDrive for brevity.

Table 4.5: Geographical summary of fastest routes for three client-locations and cloud-storage services. [Direct: solid; Detour: dashed-dotted] See also Table 4.1.

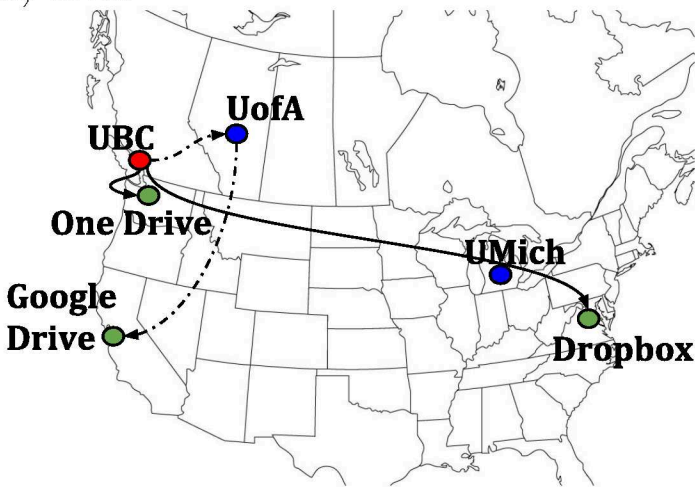
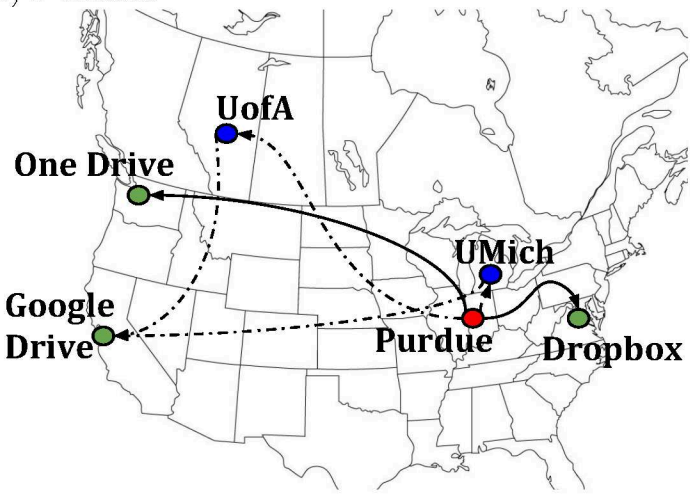
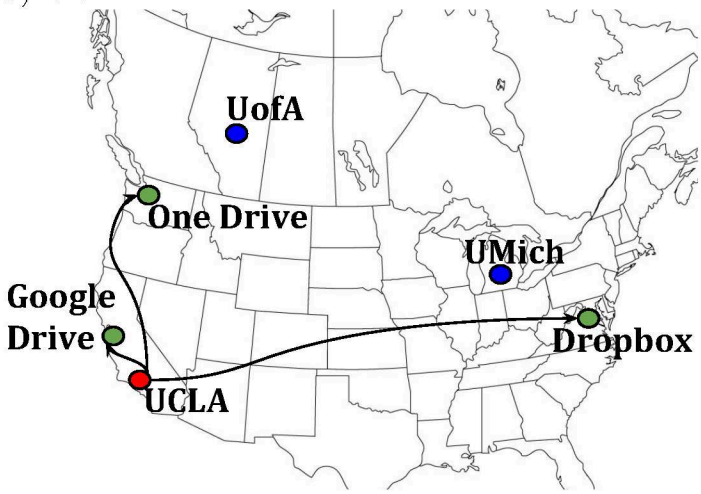
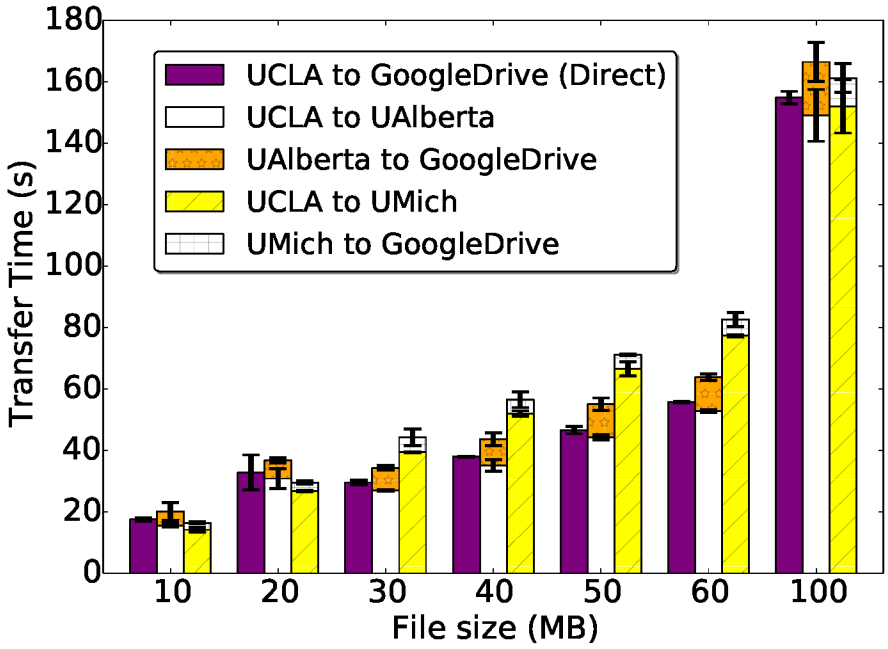
(A) **UBC**: 	Uploading to Google Drive via a detour to UofA (dotted) is faster. Uploading directly to Dropbox and OneDrive are faster than taking any detours.
(B) **Purdue**: 	Uploading to Google Drive via a detour to UofA or UMich is faster than taking direct route. Uploading directly to Dropbox and OneDrive are faster than taking any detours.
(C) **UCLA**: 	Uploading to any cloud-storage services through direct routes is faster than taking any detours

Figure 4.8: Upload performance from **UCLA to Google Drive** (direct routes and detours) [Error bar: One Standard deviation].



4.4 Discussion

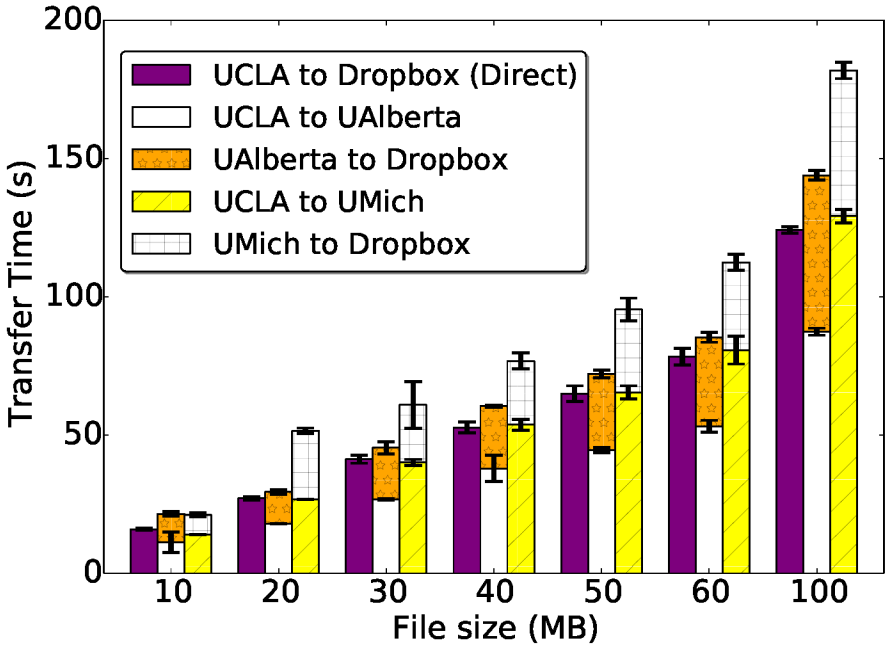
Ideally, we would like to be able to determine the root causes for each of the routing inefficiencies that we have identified. However, the goal of this paper is mainly to catalog and ameliorate the performance problems found. It has been suggested that routing table monitoring systems such as RouteViews (http://www.routeviews.org) might assist in our understanding. Certainly, RouteViews is more sophisticated than our current use of `traceroute`. But, we wanted to understand the potential performance benefits first, as motivation for further investigation.

For future work, systems like RouteViews [2] and dynamic network monitoring tools can be used as important input for a full-fledged overlay network that moves data efficiently to cloud-storage providers, based on routing data and dynamic network conditions (e.g., current congestion points).

To expand the discussion even further: the root cause of performance problems on such a dynamic, multi-administrative domain, and heterogeneous en-

[2] http://www.routeviews.org/

Figure 4.9: Upload performance from **UCLA to Dropbox** (direct routes and detours).



tity such as the Internet can be hard to prove to one's satisfaction. For example, our simple `traceroute` probing (Section 4.1) showed that Google Drive traffic from UofA took a very similar path between Vancouver and Seattle as traffic from PlanetLab UBC to Google Drive, except for one hop (i.e., pacificwave vs. other). We can see the difference in route, but so what? Is it a physical bandwidth difference at that hop? Is it a rate-limited bandwidth issue? Is it purely a PlanetLab artifact? Is it a software configuration issue affecting only that one hop? Even if we could inspect pacificwave (which is outside of our control), what would we systematically test first? For our experiments, we decided that fixing the performance problem was our first priority, working from top-down.

Apart from trying out two intermediate nodes, we also enabled optimizations in the intermediate nodes. These optimizations show the potentials of using the intermediate nodes or the DTNs. Cut-through routing is one such example of optimizations which improve the throughput in most of the cases. We will now explain the results of enabling cut-through routing in the intermediate nodes in detoured path.

4.5 Cut-through Routing Results

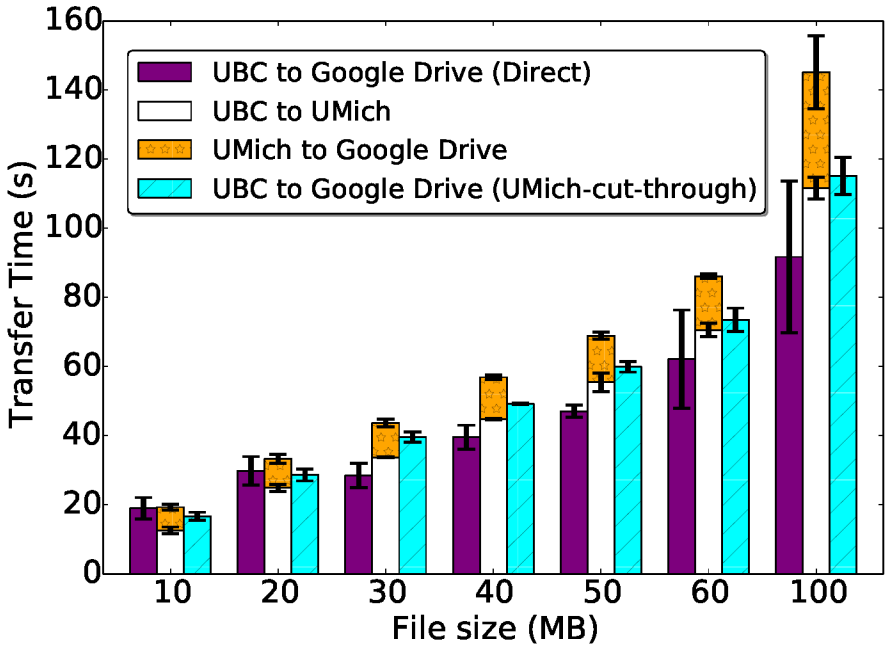
Cut-through routing has been already explained in Sec. 3.3. In this section, we discuss the results of cut-through routing for every client location and intermediate node locations. As desired, in most of the cases, cut-through routing decreases the total transfer-time from simple detour routing time, if a detour through an intermediate node is taken. However, the cut-through detour routing may not always be the best choice.

It is worth pointing out that some of the results in this section are similar to what we have seen in the previous sections in this chapter, but the experiments were done in different time-frames. For example, we can compare the Figures 4.1, 4.10 and 4.11. In Fig. 4.1, the timings for UBC-to-Google Drive via UMich (third bars) are taller than direct uploading time (first bars). This is similar to what has been observed in Fig. 4.10 where second bars (representing the simple detour of UBC-to-Google Drive via UMich) are taller than first bars (representing direct route). Similarly, in Fig. 4.1, the timings for UBC-to-Google Drive via UofA (second bars) are smaller than direct uploading time (first bars), and in Fig. 4.11 as well, second bars (representing the simple detour of UBC-to-Google Drive via UofA) is smaller than first bars (representing direct route).

Fig. 4.10 and 4.11 demonstrate the comparison of performances from UBC to Google Drive server for detours via UMich and UofA respectively, along with their results for cut-through routing. These figures depict both the scenario where cut-through routing can improve the performance and where it cannot, compared to the throughput of direct route. To be precise, in Fig. 4.10, the third bars are taller than the first bars for all file-sizes, and in Fig. 4.11, the third bars are smaller than the first bars.

We now explain individual cases further. For the case of UBC-to-Google Drive with a detour via UMich, the detour (second bars in Fig. 4.10) is not beneficial, and takes higher time than direct uploading time (first bars in the figure) for all file-sizes. However, employing cut-through routing mechanism here decreases the transfer-time in a such a way that now transfer-time for cut-

Figure 4.10: Cut-through Routing Results for **UBC to Google Drive via UMich** (February - March 2016) [Error bars: One Standard deviation] [Direct < Cut-through < Simple Detour].



through becomes nearly comparable to the direct uploading time. For example, direct route, only detour and detour with cut-through take respectively 62s, 86s and 73s to upload the 60 MB file.

Fig. 4.11 exemplifies the case where a detour with cut-through routing can improve the performance further from what a detour has already done, compared to direct route. A detour with an intermediate node at UofA, for UBC-to-Google Drive, increases the throughput for all six files in our test set. Table 4.6 shows the percentage of increment/decrement in transfer-time for only detours and detours with cut-through. For a 100 MB file, while simple detour via UofA decreases the transfer time by 38.9%, detour with cut-through reduces the time by nearly 67.9%. Detour with cut-through here causes a significant improvement, even from advantage of the detour routing, it is true for all other file-sizes in Table 4.6. Additionally, cut-through certainly improves the throughput for all the cases in Table 4.6 (4th and 5th Column) for UMich-detour, but it does not result into an overall improvement from the transfer-times of direct route (except for 20 MB), as discussed earlier.

Figure 4.11: Cut-through Routing Results for **UBC to Google Drive via UofA** (February - March 2016) [Error bars: One Standard deviation] [Cut-through < Simple Detour < Direct].

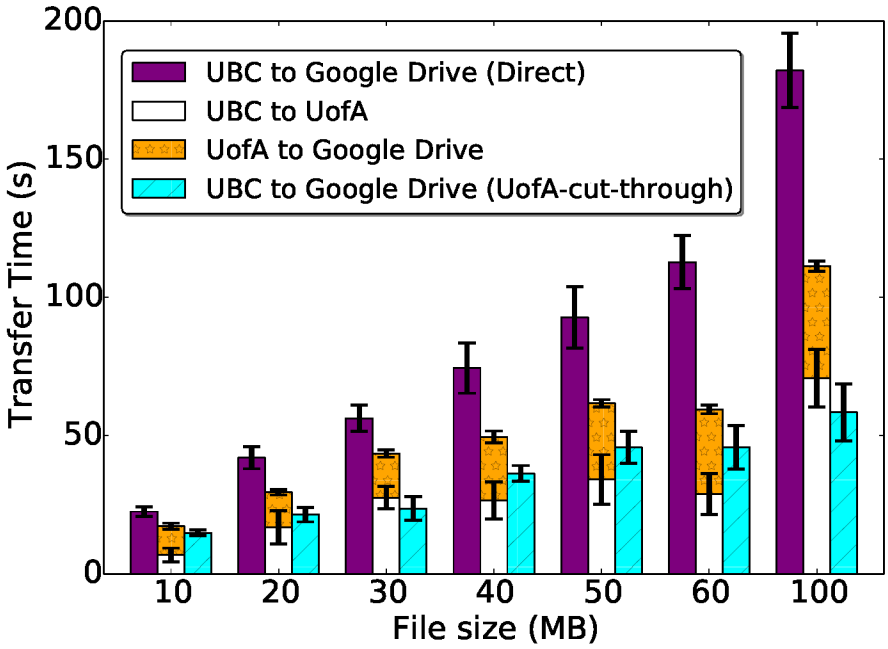


Fig. 4.12 and 4.13 present transfer-times of UBC-to-Dropbox for direct, only detours and detours with cut-through. These detours do not improve the transfer-time, compared to the transfer-times for direct routes, for all file sizes. However, like UBC-to-Google Drive via UMich, UBC-to-Dropbox via UofA becomes comparable to direct route when cut-through routing is enabled. For example, direct route, only detour and detour with cut-through take respectively 40s, 54s and 42s to upload the 60 MB file.

It might be intuitive but it is worth mentioning that the transfer-time for cut-through routing is bounded by the slowest time between the two paths that form the detour route. To explain this clearly, we show the mean transfer time for UBC-to-Dropbox detours, along with its cut-through routing time in Table 4.7. This table shows that the slow link between UofA to Dropbox affect the cut-through routing time, and the cut-through transfer times are very near to the timings of UofA to Dropbox transfer times. As an example, we can look at the file transfer times for 40 MB file. UBC to UofA takes only 8s. UofA to Dropbox takes nearly 4 times, 31s. Therefore, the whole detour takes 39s.

Table 4.6: UBC-to-Google Drive Percentage of Decrement in Transfer time of Only Detour and Detour with Cut-through over Direct route (February - March 2016).

File size (MB)	**Only Detour (UofA) %**	**Detour with Cut-through (UofA) %**	**Only Detour (UMich) %**	**Detour with Cut-through (UMich) %**
10	-23.5	-34.2	+1.6	-12.3
20	-29.7	-49.0	+11.6	-4.1
30	-22.7	-58.1	+53.4	+39.1
40	-33.5	-51.2	+43.8	+24.3
50	-33.5	-50.7	+46.5	+27.5
60	-47.2	-59.3	+38.6	+18.3
100	-38.9	-67.9	+58.3	+25.5

Table 4.7: UBC-to-Dropbox via UofA transfer times (January - February 2016).

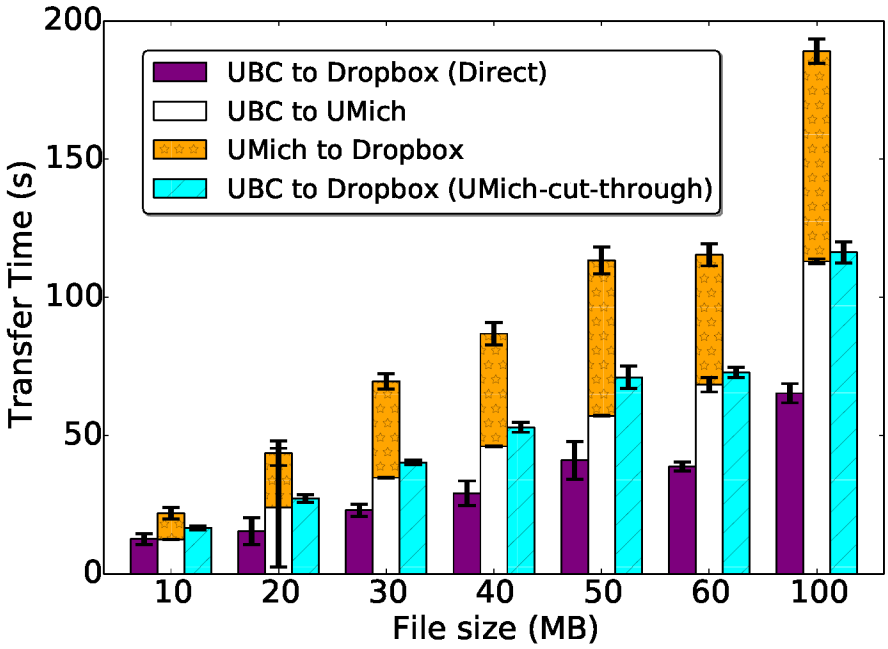
File size (MB)	**UBC to UofA (s)**	**UofA to Dropbox (s)**	**UofA-Detour + Cut-through (s)**
10	2.4	13.3	18.0
20	5.3	16.3	19.8
30	5.8	26.2	28.5
40	8.0	31.3	33.8
50	11.7	39.6	41.0
60	13.2	40.4	42.1
100	14.2	60.2	64.6

When cut-through routing is enabled, detour takes almost 34s which is closer to the time to transfer the file from UofA to Dropbox (31s). Therefore we can conclude that the slowest link affects the cut-through routing's transfer time.

Somewhat similar result can be observed for UMich-detour in Table 4.8. However, for UMich-detour, both of the intermediate links, UBC-to-UMich and UMich-to-Dropbox have comparable transfer times. Hence, cut-through can effectively decrease the transfer times of simple detours, nearly by 50%. Although 50% improvement might look significant, it is still not comparable to direct route timings, as it can be observed in Fig. 4.12. Therefore, cut-through routing can only improve the performance to a certain extent, limited by the slowest link in a path.

Fig. 4.14 and 4.15 show the results of cut-through routing for OneDrive

Figure 4.12: Cut-through Routing Results for **UBC to Dropbox via UMich** (January - February 2016) [Error bars: One Standard deviation] [Direct < Cut-through < Simple Detour].

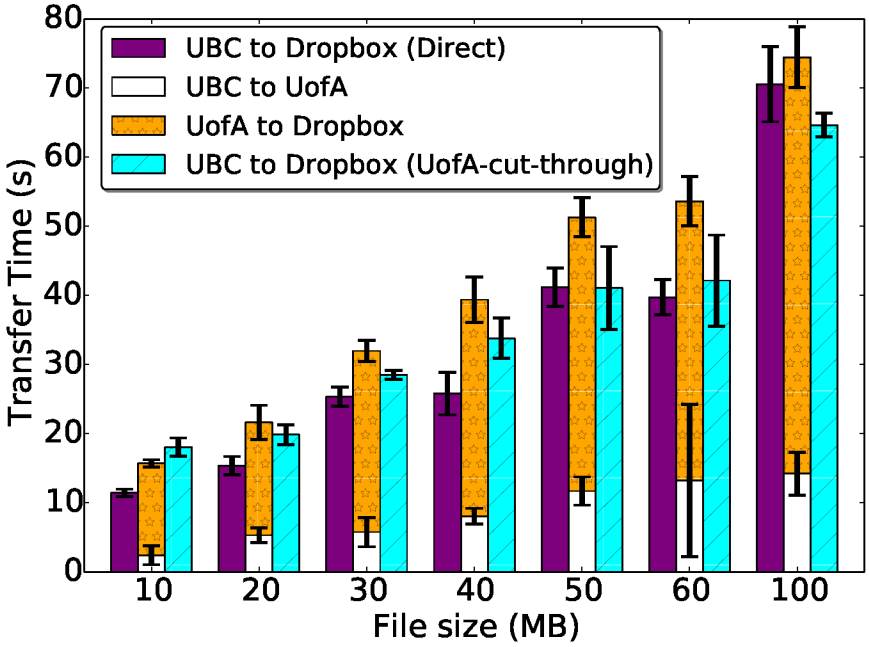


from UBC, respectively for UMich- and UofA-detour. UBC-to-OneDrive via UofA's performance for cut-through is similar to UBC-to-Dropbox via UMich, in the sense that in both the cases, cut-through reduces the total time for a detoured upload. Also, cut-through timing is limited by the slowest link, starting from the client-node to the intermediate node.

Interestingly, Fig. 4.15 for UBC-to-OneDrive via UofA is a surprising result. In this case, cut-through mechanism increases the transfer time from the timings of simple detour, except for 100 MB. We investigated the reason behind this kind of unexpected result. Apart from noticing that the mean timings with error bars of 1 standard deviation overlap with each other for only detour and detour with cut-through, we currently do not have any other explanation of this phenomenon. From the other results, we can ascertain the fact that there is no bottleneck in our code which can cause this behavior. Therefore, this scenario might be very particular to the path from UBC to OneDrive server via UofA.

As we can understand from the above discussion, the cut-through results

Figure 4.13: Cut-through Routing Results for **UBC to Dropbox via UofA** (January - February 2016) [Error bars: One Standard deviation] [Direct $\sim$ Cut-through $<$ Simple Detour].



can be put into a few categories. Table 4.9 and 4.10 represent the categorization for UBC, Purdue and UCLA client locations, and Google Drive, Dropbox, OneDrive cloud-storage services by two aspects. Table 4.9 shows us the categorization based the performance of cut-through routing when compared with the timings of direct route and simple detour. Corresponding figure numbers are also inscribed in the table. Moreover, Table 4.10 puts the results into categories based on the dependence of the cut-through timings on the slowest links. For example, timings for UBC-to-Google-Drive via UMich is capped by the slowest link from UBC-to-UMich, and not UMich-to-Google-Drive. On the contrary, UBC-to-Dropbox via UofA is capped by UofA-to-Dropbox's slow performance. In Table 4.10, the indecisive column signifies those scenarios where cut-through timings show abnormal behavior (such as UBC-to-OneDrive via UofA). Proportional to both of the intermediate links means that the transfer times of both of the paths in a detour are comparable and have similar effect on the timings of cut-through routing. The figures for all other locations, intermediate nodes and cloud-storage services have been included for completeness

Table 4.8: UBC-to-Dropbox via UMich transfer times (January - February 2016).

File size (MB)	**UBC to UMich (s)**	**UMich to Dropbox (s)**	**UMich-Detour + Cut-through (s)**
10	12.4	9.5	16.6
20	23.9	19.7	27.2
30	34.8	34.8	40.2
40	46.1	40.8	52.9
50	57.2	56.2	71.1
60	68.4	47.0	72.8
100	113.0	76.0	116.3

in Fig. 4.16, 4.17, 4.18, 4.19, 4.20, 4.21, 4.22, 4.23, 4.24, 4.25, 4.26, 4.27.

Table 4.9: Classification of cut-through routing timings based on its comparative performance with direct and simple detour timings

Client location	**UBC**	**Purdue**	**UCLA**
Direct < Cut-through < Simple Detour	Google Drive(via UMich)-F. 4.10, Dropbox(via UMich)-F. 4.12, OneDrive(via UMich)-F. 4.14		Google Drive(via UMich)-F. 4.22
Cut-through < Simple Detour < Direct	Google Drive(via UofA)-F. 4.11		
Cut-through < Direct < Simple Detour			Google Drive(via UofA)-F. 4.23
Direct < Simple Detour ∼ Cut-through			OneDrive(via UofA)-F. 4.27

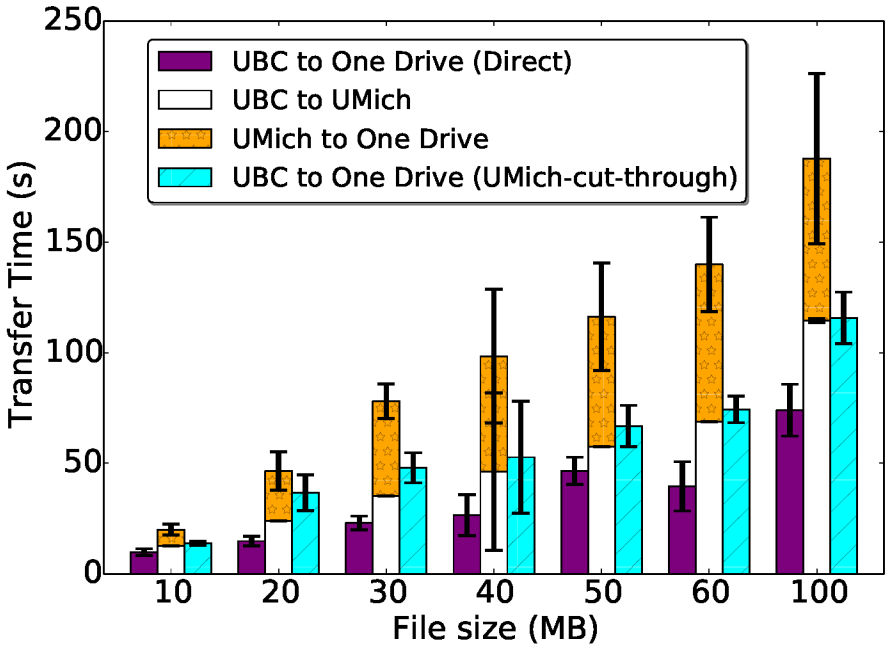
Direct ∼ Cut-through < Simple Detour	Dropbox(via UofA)-F. 4.13		Dropbox(via UMich)-F. 4.24, Dropbox(via UofA)-F. 4.25, OneDrive(via UMich)-F. 4.26
Cut-through ∼ Simple Detour < Direct		Google Drive(via UMich)-F. 4.16, Google Drive(via UofA)-F. 4.17	
Cut-through ∼ Simple Detour ∼ Direct	OneDrive(via UofA)-F. 4.15	Dropbox(via UMich)-F. 4.18, Dropbox(via UofA)-F. 4.19, OneDrive(via UMich)-F. 4.20, OneDrive(via UofA)-F. 4.21	

4.6 Detour Routing Statistics on PlanetLab Nodes

We have extensively studied 3 PlanetLab client locations (UBC, Purdue, UCLA) and 2 intermediate nodes (UMich - PlanetLab, UofA cluster) in the previous sections. However, we wanted to get a brief overview of the inefficiencies in PlanetLab. Therefore, we have also experimented with 20 PlanetLab nodes to find out probable beneficial detours in the platform in North America. This experimentation also helps us pointing out that these types of efficiencies are not only constrained to our selected 3 locations (i.e - UBC, UCLA, Purdue). Rather, such anomalies do exist pervasively in a popular experimentation platform like PlanetLab.

It is worth noting that these experimentations with 20 PlanetLab nodes are not as extensive as the other experiments discussed in previous sections. We only experimented with a randomly generated 10 MB file. That single file has been uploaded from the client locations directly and through detours, to the cloud-storage servers. Out of total 7 runs, average of the last 5 runs is

Figure 4.14: Cut-through Routing Results for **UBC to OneDrive via UMich** (Feb-March 2016) [Error bar: One Standard deviation] [Direct < Cut-through < Simple Detour].



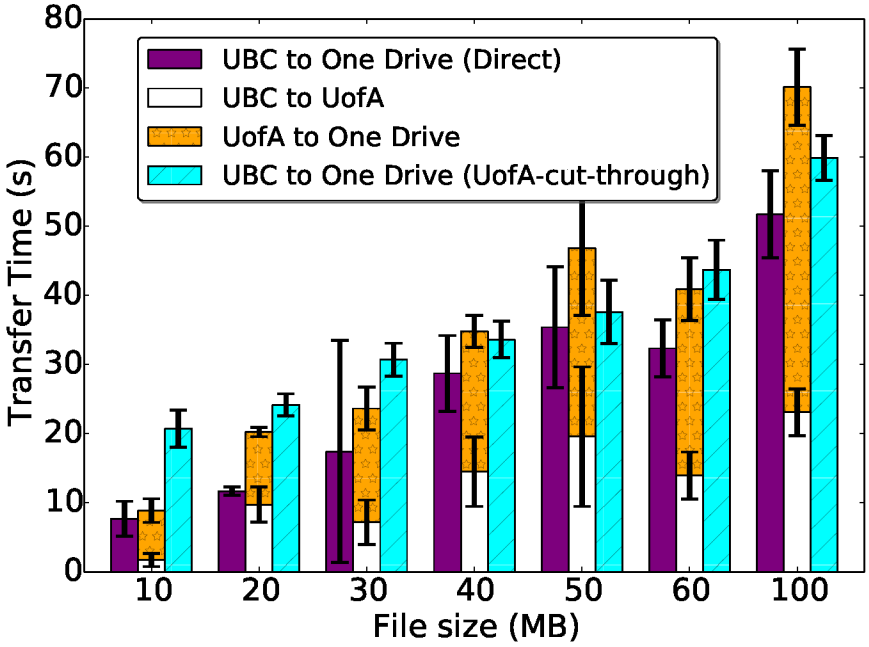
measured.

Our experimented 20 PlanetLab nodes are following:

University of British Columbia (UBC), University of Michigan (UMich), Purdue University, University of California, Los Angeles (UCLA), Rice University, Rutgers University, University of Oregon (UOregon), University of Wisconsin Madison (UWM), Boston University, Princeton University, Oklahoma State University (OkState), Brown University, University of Texas, Arlington (UTA), University of New Mexico (UNM), University of Washington (UWash), University of Nebraska (UNebraska), University of Maryland, College Park (UMCP), University of Pittsburgh (UPitt), Indiana University, Bloomington (IUB), University of South Florida (USF).

Table 4.11 summarizes the locations of beneficial intermediate nodes for those client nodes, for which any improvement in throughput has been observed. There are total 6, 10 and 10 PlanetLab nodes which got benefited respectively for Google Drive, Dropbox and OneDrive. Fig. 4.32 depicts the same information in graphical form for 20 PlanetLab nodes and 3 cloud-storage

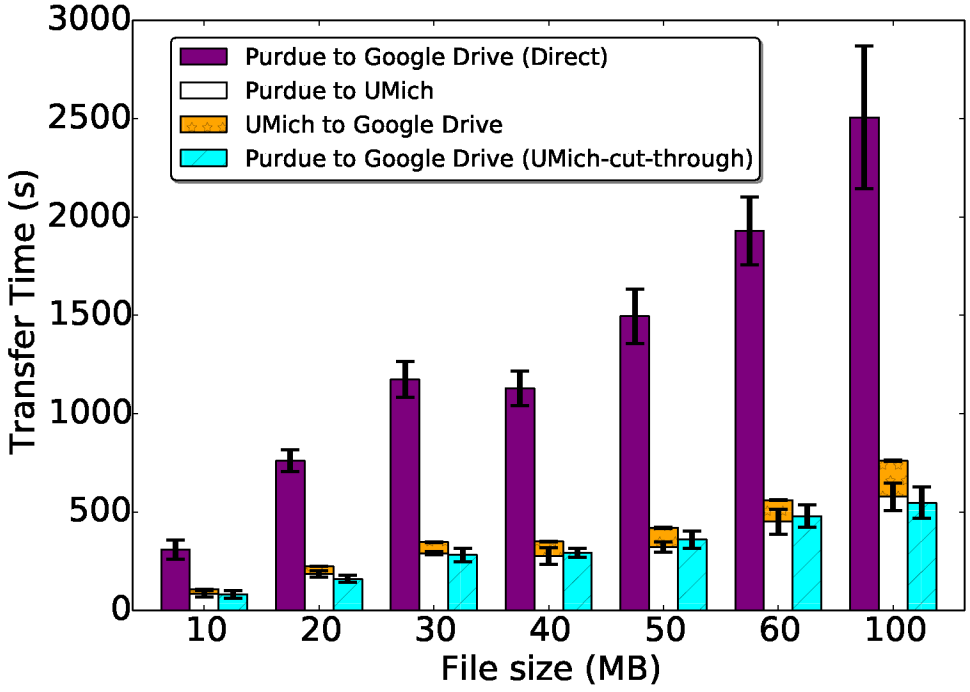
Figure 4.15: Cut-through Routing Results for **UBC to OneDrive via UofA** (Feb-March 2016) [Error bar: One Standard deviation] [Direct < Simple Detour ∼ Cut-through].



services. We will now mention some of the observations based on these experimentations.

In the previous sections, we have already discussed about Purdue. However, those experiments involve only two intermediate nodes which are UMich PlanetLab node and UofA non-PlanetLab node. Using the other 19 PlanetLab nodes as the intermediate nodes for Purdue-to-cloud-storage servers reveals some more interesting facts. For Google Drive and OneDrive, simple detour with all other intermediate nodes improves throughput for Purdue with the randomly generated 10 MB file, according to Table 4.11. Poor performance of Purdue can be explained by exploring Fig. 4.28 which summarizes the timings to upload the randomly generated 10 MB file to Google Drive from 20 PlanetLab locations. Fig. 4.28 shows how Purdue (third data-point from left-hand-side) is far off and higher than other PlanetLab locations. Additionally, Fig. 4.29 shows the timings to upload the same 10 MB file from Purdue to other PlanetLab nodes and to the cloud-storage servers. In Fig. 4.29, it is clear that Purdue-to-Google Drive (third data-point from right hand side) takes much

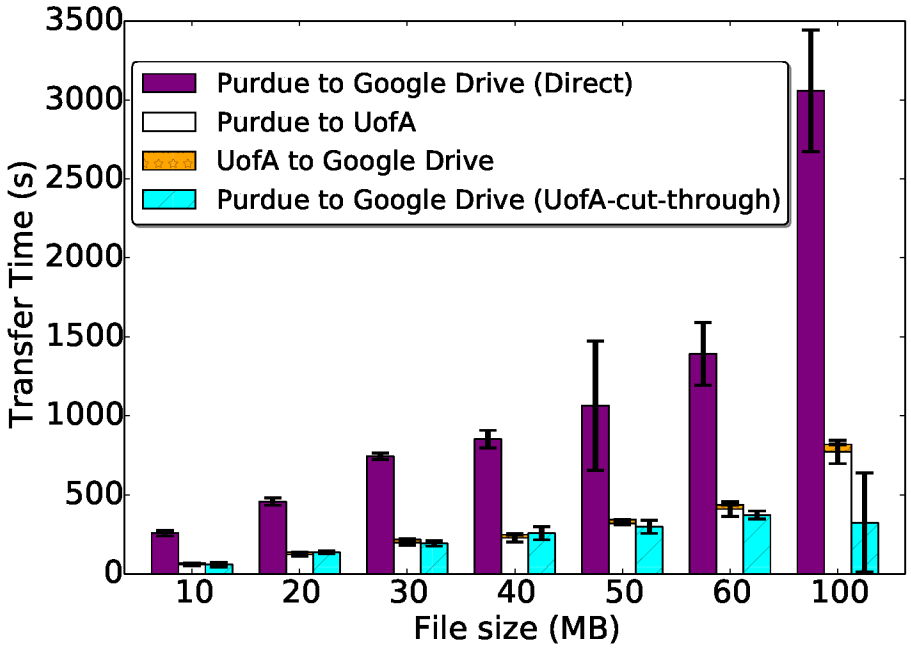
Figure 4.16: Cut-through Routing Results for **Purdue to Google Drive via UMich** (February - March 2016) [Error bar: One Standard deviation] [Cut-through $\sim$ Simple Detour $<$ Direct].



higher time than timing to transfer to the other PlanetLab nodes. Therefore, Purdue's performance for Google Drive is improved by using detours. Similar case is observed for Purdue-to-OneDrive.

There are other cases of poor performances which can be observed in Table 4.11. For example, University of Pittsburgh has poor performance for Google Drive. This is evident in Fig. 4.28, where its data-point (third from right-hand-side) is slightly off from the data-points of other PlanetLab locations. Furthermore, Fig. 4.30 shows timings to upload a 10 MB file from 20 PlanetLab locations to Dropbox server. Apart from Purdue, data-points of Boston University, University of New Mexico (UNM) and University of South Florida (USF) are higher than others. Their higher timings corroborate to the higher number of beneficial intermediate nodes for Boston (5), UNM (9) and USF (11) in Table 4.11. Of course, the benefit from a particular PlanetLab node is dependent on the transfer time between the particular PlanetLab node and other PlanetLab nodes. For completeness, Fig. 4.31 depicting the timings

Figure 4.17: Cut-through Routing Results for **Purdue to Google Drive via UofA** (February - March 2016) [Error bar: One Standard deviation] [Cut-through $\sim$ Simple Detour $<$ Direct].



to upload a 10 MB file from 20 PlanetLab locations to OneDrive, is included.

Table 4.11: List of benefited client node locations among 20 PlanetLab nodes with their beneficial intermediate nodes for different cloud-storage providers using simple detour routing. [See Fig. 4.32 for graphical representation of number of detours for each PlanetLab location].

Client-node location	**Google Drive**	**Dropbox**	**OneDrive**
UBC	UOregon		
Purdue	UBC, UMich, UCLA, Rice, Rutgers, UOregon, UWM, Boston, Princeton, OkState, Brown, UTA, UNM, UWash, UNebraska, UMCP, UPitt, IUB, USF	UBC, Rice, Rutgers, UOregon, Boston, Princeton, Brown, UTA, UWash, UNebraska, UMCP, UPitt, IUB	UBC, UMich, UCLA, Rice, Rutgers, UOregon, UWM, Boston, Princeton, OkState, Brown, UTA, UNM, UWash, UNebraska, UMCP, UPitt, IUB, USF

UCLA			OkState, UTA, UNebraska
UMich		UMCP	
Rutgers	UMich, UWM, Princeton, OkState, UTA, UMCP, USF	UMCP	UMich, Princeton
UWash	UOregon	UBC	UBC
UPitt	UBC, UMich, UCLA, Rice, Rutgers, UOregon, UWM, Boston, Princeton, OkState, Brown, UTA, UNM, UWash, UNebraska, UMCP, IUB, USF		Princeton, OkState
IUB	UMich, UWM, OkState		UMich, UWM, Princeton, OkState, UTA, UNebraska, UMCP, USF
Rice		UMCP	OkState
Boston		Rutgers, Princeton, Brown, UMCP, UPitt	UMich, Rice, Rutgers, UWM, Princeton, OkState, Brown, UTA, UNebraska, UMCP, UPitt, IUB, USF
OkState		UNebraska	
UTA		UNebraska, UMCP	
UNM		UBC, UMich, Rutgers, UWM, OkState, Brown, UWash, UNebraska, UMCP	
UOregon			UBC

USF		UBC, UMich, Rice, Rutgers, UWM, OkState, Brown, UTA, UNebraska, UMCP, IUB	
Brown			OkState

Figure 4.18: Cut-through Routing Results for **Purdue to Dropbox via UMich** (January - February 2016) [Error bar: One Standard deviation] [Cut-through ∼ Simple Detour ∼ Direct].

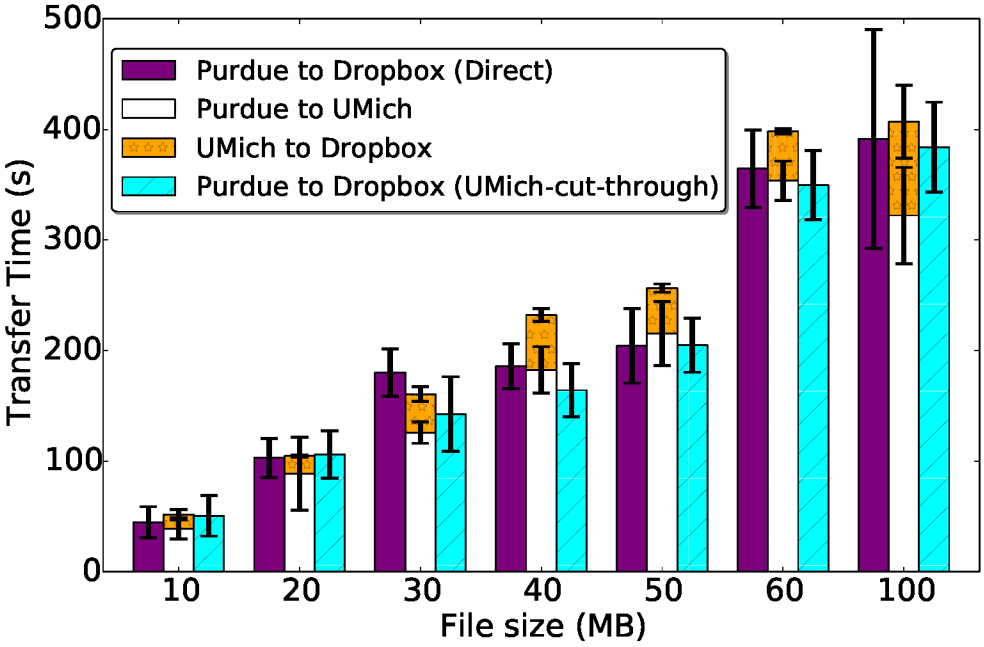


Figure 4.19: Cut-through Routing Results for **Purdue to Dropbox via UofA** (January - February 2016) [Error bar: One Standard deviation] [Cut-through ∼ Simple Detour ∼ Direct].

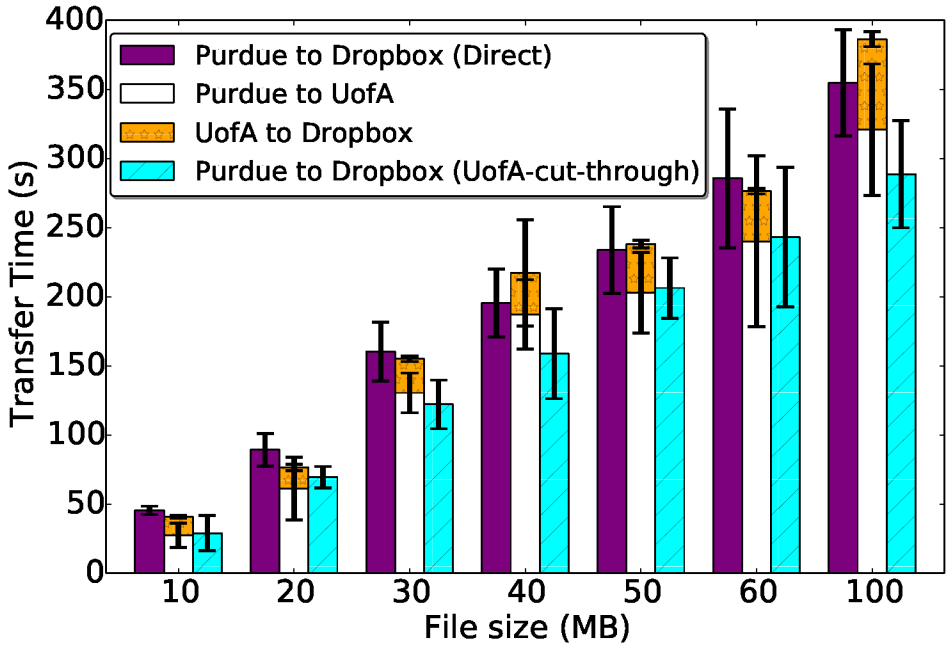


Table 4.10: Classification of Cut-through timings based on the slowest links among **client-to-intermediate-node (C2I)** and **intermediate-node-to-cloud-storage-server (I2CS)**.

Client-node location	**Cut-through time proportional to C2I time**	**Cut-through time proportional to I2CS time**	**Cut-through time proportional to both**	**Indecisive**
UBC	Google Drive(via UMich)-F. 4.10, Dropbox(via UMich)-F. 4.12	Dropbox(via UofA)-F. 4.11	Google Drive(via UofA)-F. 4.11, OneDrive(via UMich)-F. 4.14	OneDrive(via UofA)-F. 4.15
Purdue	Google Drive(via UMich)-F. 4.16, Google Drive(via UofA)-F. 4.17, Dropbox(via UMich)-F. 4.18, Dropbox(via UofA)-F. 4.19, OneDrive(via UMich)-F. 4.20, OneDrive (via UofA)-F. 4.21			
UCLA	Google Drive(via UMich)-F. 4.22, Google Drive(via UofA)-F. 4.23, OneDrive(via UMich)-F. 4.26, OneDrive(via UofA)-F. 4.27		Dropbox(via UMich)-F. 4.24, Dropbox(via UofA)-F. 4.25	

Figure 4.20: Cut-through Routing Results for **Purdue to OneDrive via UMich** (February - March 2016) [Error bar: One Standard deviation] [Cut-through ∼ Simple Detour ∼ Direct].

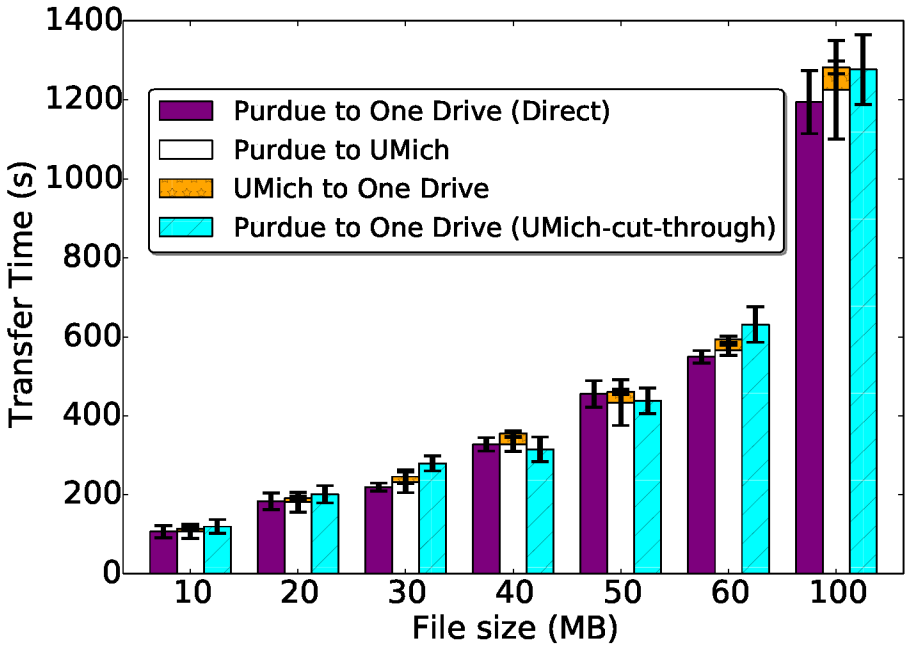


Figure 4.21: Cut-through Routing Results for **Purdue to OneDrive via UofA** (February - March 2016) [Error bar: One Standard deviation] [Cut-through ∼ Simple Detour ∼ Direct].

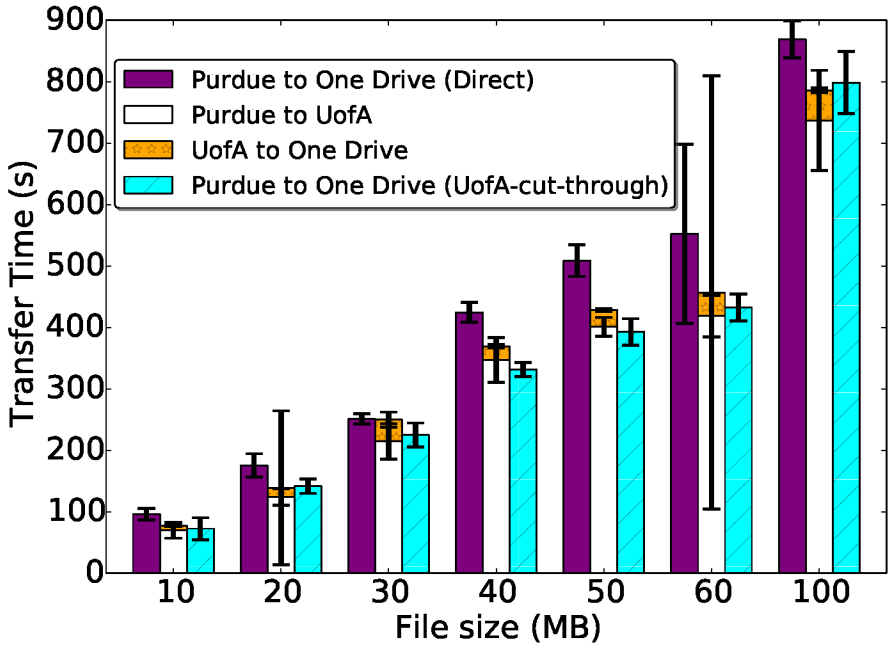


Figure 4.22: Cut-through Routing Results for **UCLA to Google Drive via UMich** (January - February 2016) [Error bar: One Standard deviation] [Direct < Cut-through < Simple Detour].

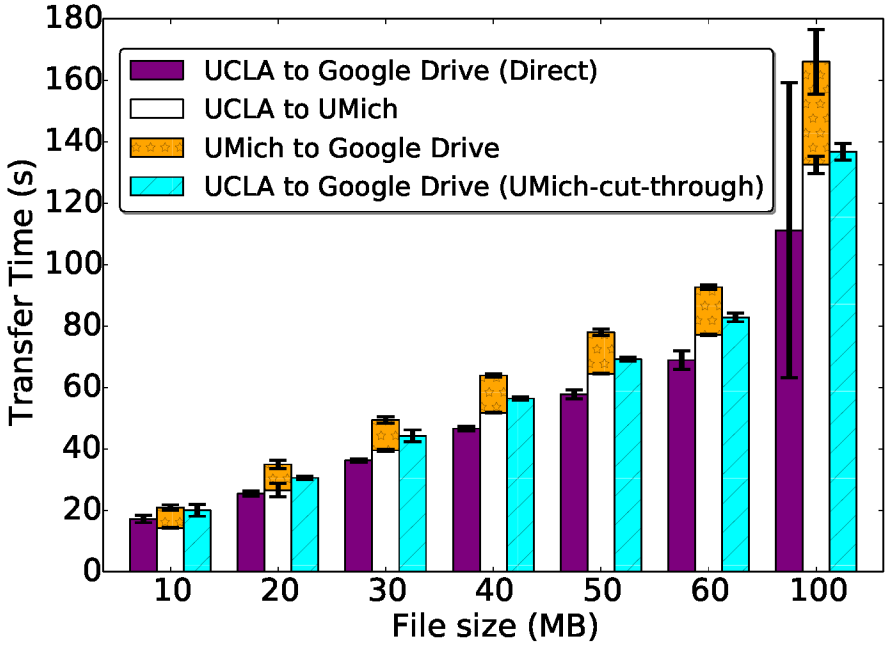


Figure 4.23: Cut-through Routing Results for **UCLA to Google Drive via UofA** (January - February 2016) [Error bar: One Standard deviation] [Cut-through < Direct < Simple Detour].

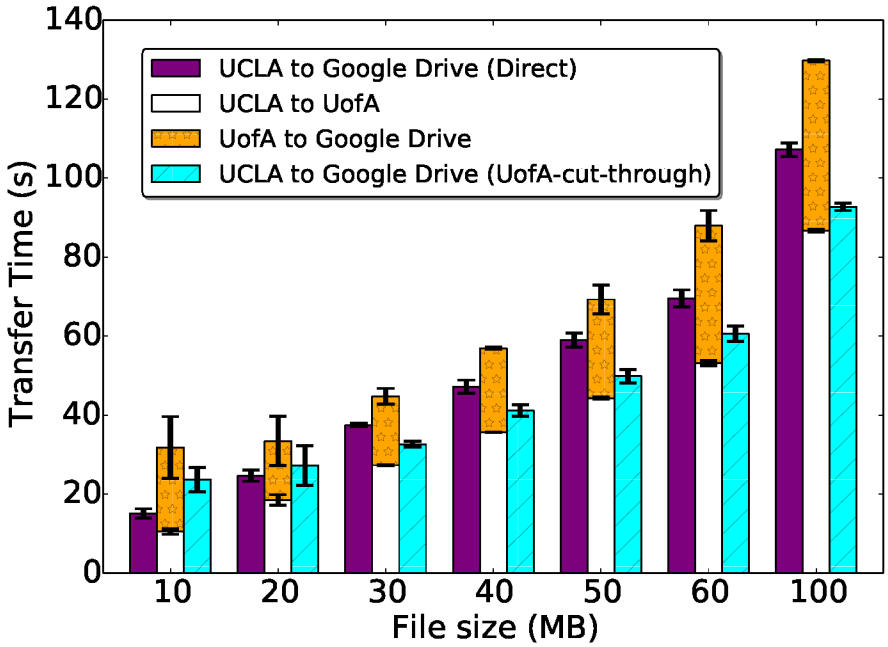


Figure 4.24: Cut-through Routing Results for **UCLA to Dropbox via UMich** (January - February 2016) [Error bar: One Standard deviation] [Direct $\sim$ Cut-through $<$ Simple Detour].

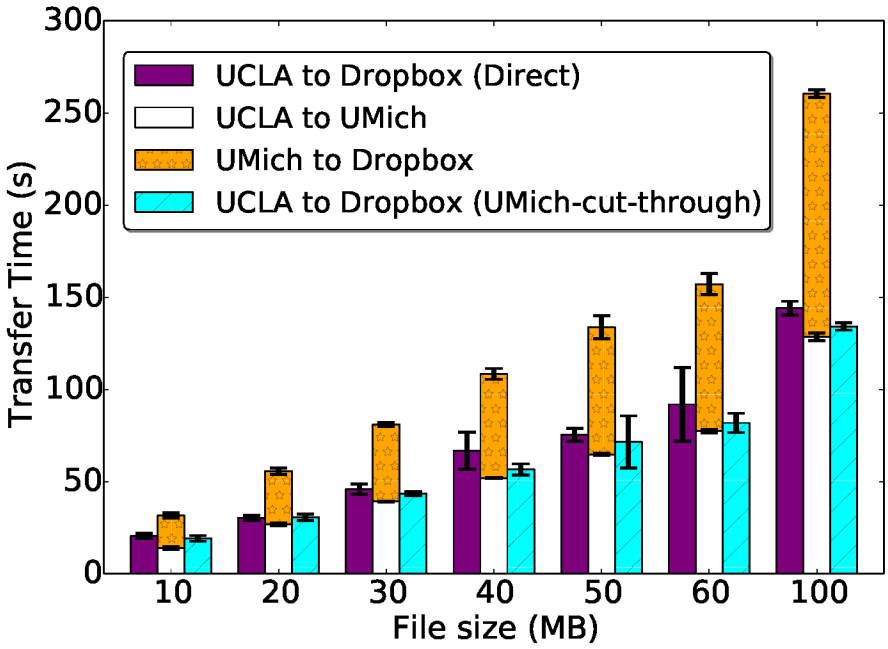


Figure 4.25: Cut-through Routing Results for **UCLA to Dropbox via UofA** (January - February 2016) [Error bar: One Standard deviation] [Direct $\sim$ Cut-through $<$ Simple Detour].

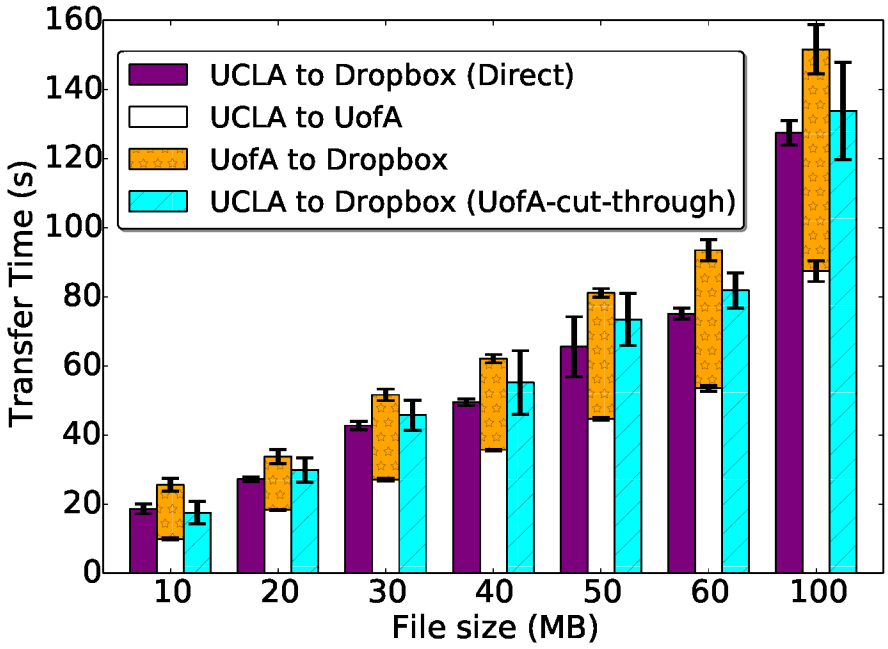


Figure 4.26: Cut-through Routing Results for **UCLA to OneDrive via UMich** (Feb - March 2016) [Error bar: One Standard deviation] [Direct $\sim$ Cut-through $<$ Simple Detour].

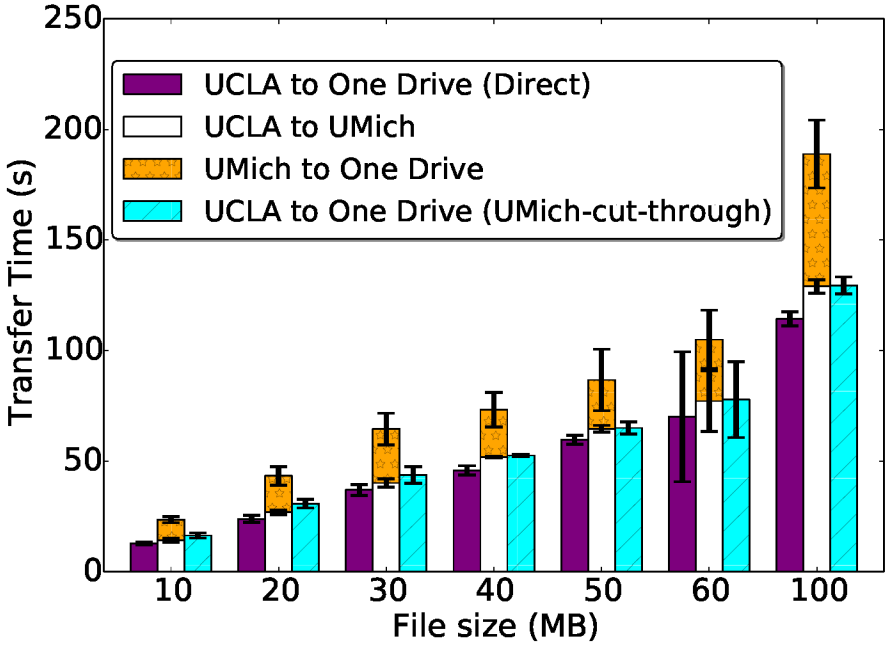


Figure 4.27: Cut-through Routing Results for **UCLA to OneDrive via UofA** (Feb - March 2016) [Error bar: One Standard deviation] [Direct $\sim$ Cut-through $\sim$ Simple Detour].

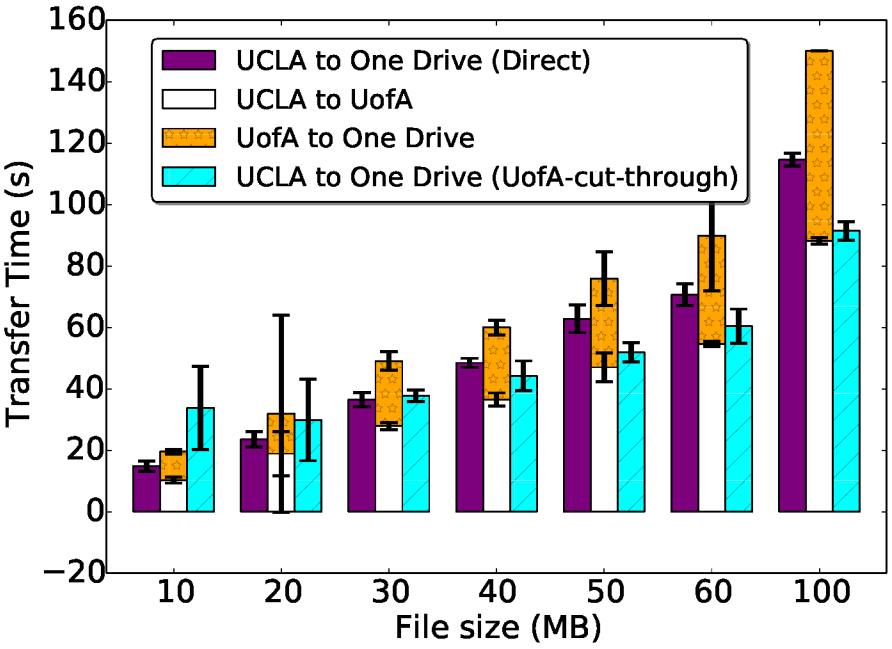


Figure 4.28: Time to upload a 10 MB file from various PlanetLab locations to Google Drive.

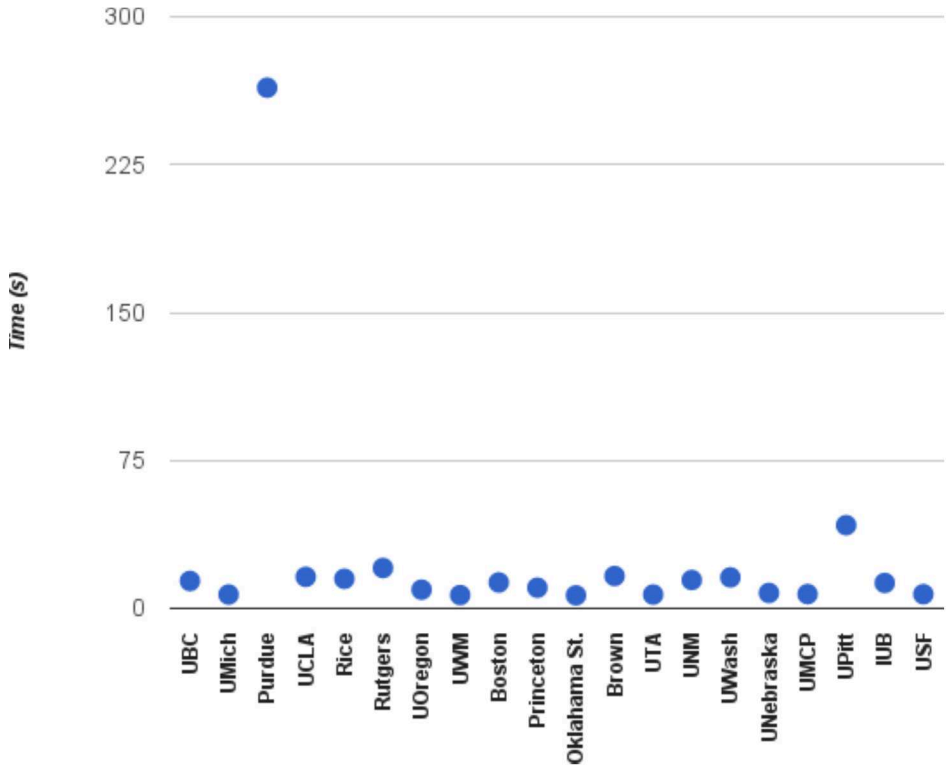


Figure 4.29: Time to transfer a 10 MB file from Purdue University PlanetLab node to other PlanetLab locations and cloud-storage servers.

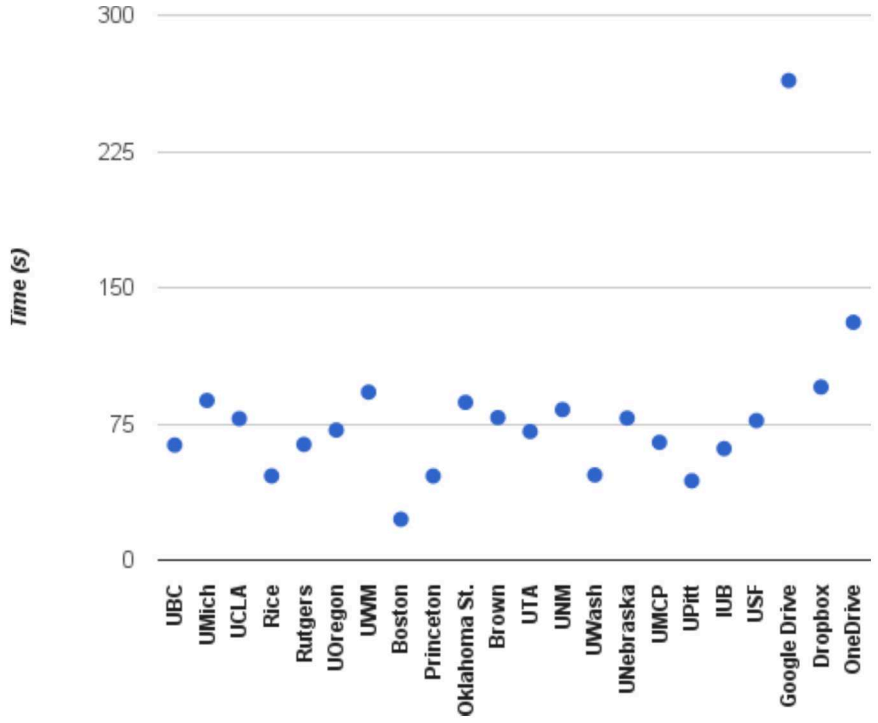


Figure 4.30: Time to upload a 10 MB file from various PlanetLab locations to Dropbox.

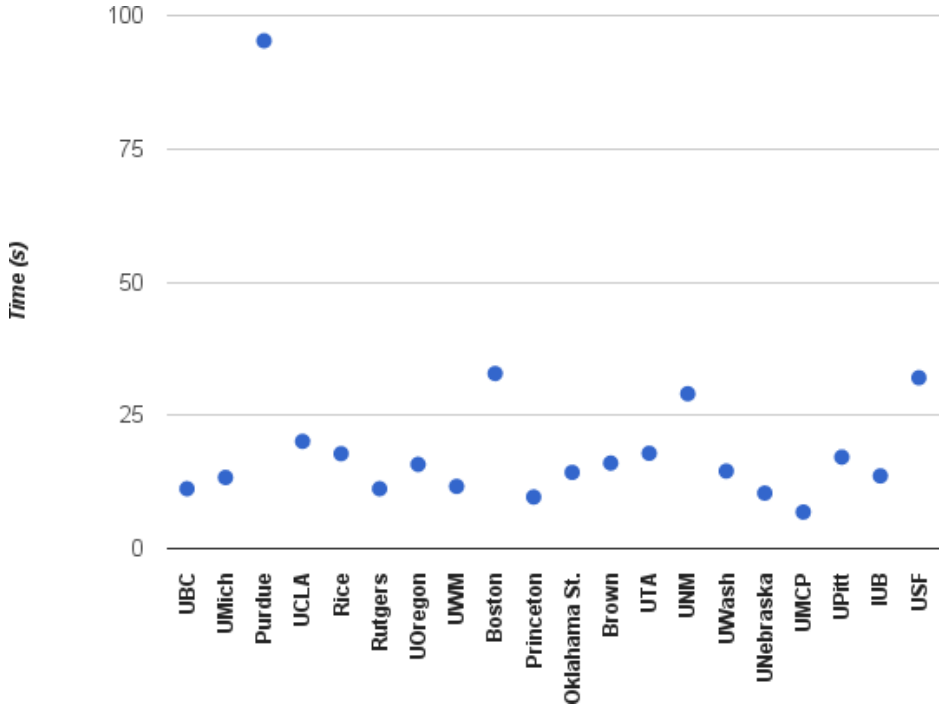


Figure 4.31: Time to upload a 10 MB file from various PlanetLab locations to OneDrive.

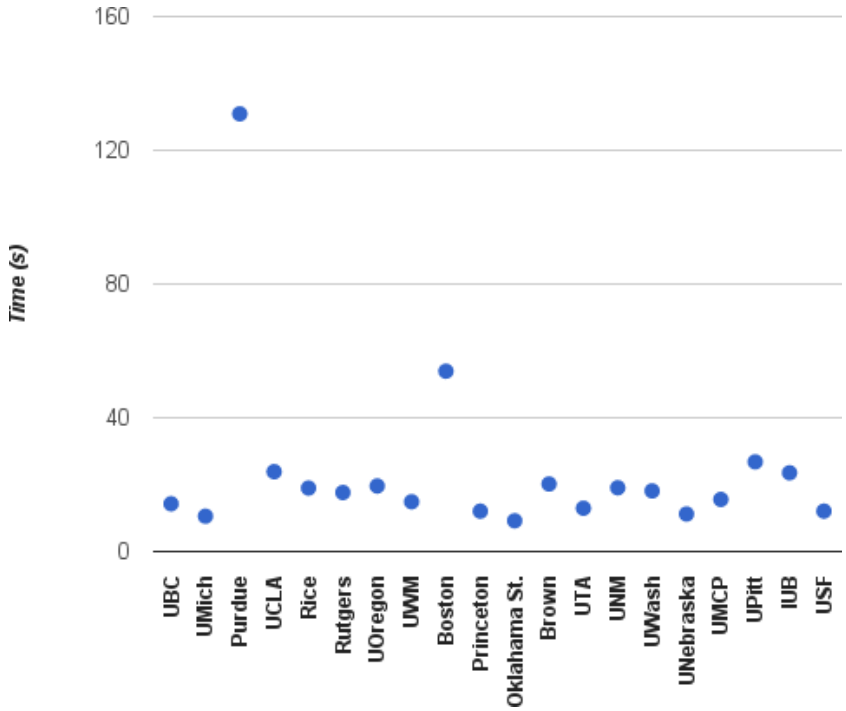
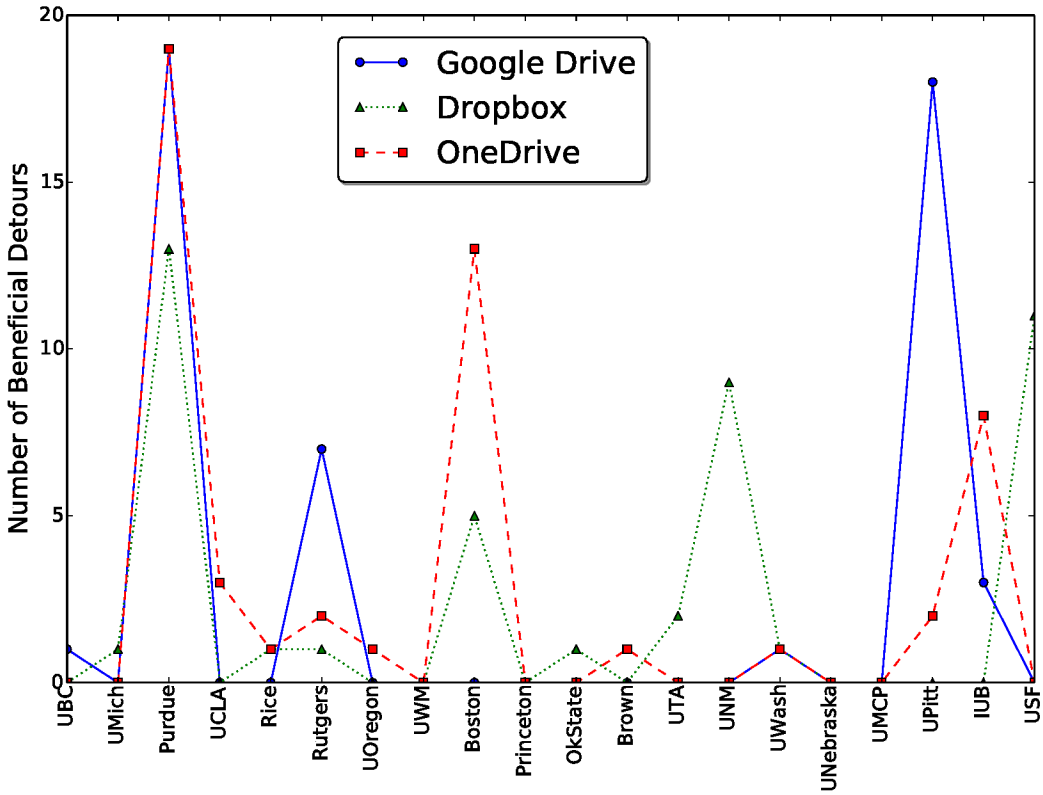


Figure 4.32: Number of beneficial detours for 20 PlanetLab locations and for all 3 cloud-storage services [See Table 4.11 for details].



Chapter 5

Summary and Conclusions

Cloud-storage services are becoming more and more popular day-by-day for their conveniences and other advantages. However, their benefits are constrained by the bandwidths of users' network connection. While normal users may not have much control over the network to maximize the throughput, large organizations may have that capability. Our presented work shows how cloud-storage services are affected by the users' network bandwidth and locations. We also describe how the large organizations with multiple points of presence can get benefits by examining their existing networking infrastructure.

As a cautionary tale and case study, we have identified some inefficiencies in how traffic to well-known cloud-storage providers can vary significantly by a factor of 5 or more, depending on the location of the source and sink of the data. Using simple *routing detours*, we show how the client-to-cloud-storage throughput can be improved by factors of over 3x, even with the detour overheads. Although the specific inefficiencies in this paper might be transitory, they do affect a growing class of traffic of data, namely traffic between clients and cloud-storage providers. As part of the larger discussion of data transfer nodes (DTNs) and overlay networks, our routing detours are simple and effective for a contemporary use case.

Overlay networking is an well-established concept where researchers want to take advantage of existing knowledge about a network. However, overlay networking should be revisited because the emergence of new network design pattern such as the Science DMZ network architecture [5]. Therefore, we

design the simple detours with DTNs and experimented on popular experimental platform, PlanetLab to show that overlay networking can be applied on cloud-storage network traffic for the opportunities of benefits.

We have also implemented one optimization (cut-through routing) inside the DTNs to further improve the throughput of client-to-cloud-storage data-transfer. Cut-through routing is able to decrease the total transfer times, sometimes by 50% from the transfer times of simple detour. Although cut-through routing may not always produce the best results (i.e - may not always be better than direct routes), it mostly does not hurt the performance of simple detours, as it can be seen in Sec. 4.5.

Cut-through routing is an example of optimizations between consecutive nodes in a network path. We apply cut-through routing in a DTN which coordinates between a client node and a cloud-storage server. However, there are other types of optimizations which can be applied in a single DTN. We discuss in Sec. 5.1 about such an optimization (caching) on which we are currently working.

Furthermore, as future work, our group plans to expand the functionality of our routing detours to deal with other bottlenecks (e.g., firewalls, like Science DMZ) and to monitor and bypass dynamic bottlenecks on the wide-area network.

5.1 Directions for Future Work

Apart from optimizations between multiple nodes, there are possibilities to improve throughput by employing optimizations inside a node, in our case the DTNs. Overall network throughput (client-to-cloud-storage throughput) can further be improved by employing caching inside the DTNs. We are currently working on caching block-level data inside the DTNs. Block-level caching is especially advantageous when the same file is being uploaded multiple times with a few modifications. In cloud-storage services, users often upload different versions of same files such as log files of a process or a research experiment. Block-level caching can be beneficial in those cases.

Another possible direction for future work is to automate the intermediate node or DTN selection process, based on certain parameters. Right now, we have pre-selected the intermediate nodes and presented a measurement study for those nodes against some client node locations. However, the selection of intermediate nodes can be made automatically based on file sizes, current network-loads, etc. If an intermediate node is chosen based on selected set of features, that might give more advantage than static or manual selection of nodes. However, the selection algorithm needs to investigate the proper set of parameters which affect the network throughput between client nodes and cloud-storage services. Finding effective parameters to the intermediate node selection algorithm and finally developing such an algorithm may encourage a larger research project.

Bibliography

[1] D. Andersen, H. Balakrishnan, M.F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *18th Symposium on Operating System Principles (SOSP)*, pages 131–145, Banff, Canada, October 2001.

[2] Ali C. Begen, Yucel Altunbasak, and Ozlem Ergun. Multi-path selection for multiple description encoded video streaming. In *Communications, 2003. ICC'03. IEEE International Conference on*, volume 3, pages 1583–1589. IEEE, 2003.

[3] A. Bergen, Y. Coady, and R. McGeer. Client bandwidth: The forgotten metric of online storage providers. In *Communications, Computers and Signal Processing (PacRim), 2011 IEEE Pacific Rim Conference on*, pages 543–548, Aug 2011.

[4] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 407–418, New York, NY, USA, 2003. ACM.

[5] Eli Dart, Lauren Rotman, Brian Tierney, Mary Hester, and Jason Zurawski. The science dmz: A network design pattern for data-intensive science. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 85:1–85:10, New York, NY, USA, 2013. ACM.

[6] Idilio Drago, Enrico Bocchi, Marco Mellia, Herman Slatman, and Aiko Pras. Benchmarking personal cloud storage. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, pages 205–212. ACM Press, 2013.

[7] Idilio Drago, Marco Mellia, Maurizio M. Munafo, Anna Sperotto, Ramin Sadre, and Aiko Pras. Inside Dropbox: Understanding Personal Cloud Storage Services. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, IMC '12, pages 481–494, New York, NY, USA, 2012. ACM.

[8] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. In *Proceedings of the 22Nd International Conference on Software Engineering*, ICSE '00, pages 407–416, New York, NY, USA, 2000. ACM.

[9] D Holman, D Lee, et al. A survey of routing techniques in store-and-forward and wormhole interconnects. *Sandia National Laboratories Albuquerque, New Mexico*, 87185, 2008.

[10] IETF. The OAuth 2.0 Authorization Framework. https://tools.ietf.org/html/rfc6749. Last accessed: 06-Mar-2016.

[11] IP Location Finder. https://www.iplocation.net/. Last accessed: 06-Mar-2016.

[12] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole, Jr. Overcast: Reliable multicasting with on overlay network. In *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4*, OSDI'00, pages 14–14, Berkeley, CA, USA, 2000. USENIX Association.

[13] Ryoichi Kawahara, Eng Keong Lua, Masato Uchida, Satoshi Kamei, and Hideaki Yoshino. On the quality of triangle inequality violation aware routing overlay architecture. In *INFOCOM 2009, IEEE*, pages 2761–2765. IEEE, 2009.

[14] Parviz Kermani and Leonard Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks (1976)*, 3(4):267–286, 1979.

[15] Zhenhua Li, Cheng Jin, Tianyin Xu, Christo Wilson, Yao Liu, Linsong Cheng, Yunhao Liu, Yafei Dai, and Zhi-Li Zhang. Towards network-level efficiency for cloud storage services. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, pages 115–128, New York, NY, USA, 2014. ACM.

[16] Eng Keong Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys Tutorials, IEEE*, 7(2):72–93, Second 2005.

[17] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 161–172, New York, NY, USA, 2001. ACM.

[18] Soham Sinha, Di Niu, Zhi Wang, and Paul Lu. Mitigating routing inefficiencies to cloud-storage providers: A case study. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2016 IEEE International*. IEEE, 2016.

[19] D.A. Tran, K.A. Hua, and T. Do. Zigzag: an efficient peer-to-peer scheme for media streaming. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 1283–1292 vol.2, March 2003.

[20] Haiyang Wang, Ryan Shea, Feng Wang, and Jiangchuan Liu. On the Impact of Virtualization on Dropbox-like Cloud File Storage/Synchronization Services. In *IEEE Workshop on Quality of Service*, IWQoS '12, pages 11:1–11:9, Piscataway, NJ, USA, 2012. IEEE Press.

[21] Bo Zhang, T.S.E. Ng, A. Nandi, Rudolf H. Riedi, P. Druschel, and Guohui Wang. Measurement-based analysis, modeling, and synthesis of the internet delay space. *Networking, IEEE/ACM Transactions on*, 18(1):229–242, Feb 2010.

[22] Xinyan Zhang, Jiangchuan Liu, Bo Li, and T.P. Yum. Coolstreaming/-donet: a data-driven overlay network for peer-to-peer live media streaming. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 2102–2111 vol. 3, March 2005.

[23] Han Zheng, Eng Keong Lua, Marcelo Pias, and Timothy G Griffin. Internet routing policies and round-trip-times. In *Passive and Active Network Measurement*, pages 236–250. Springer, 2005.