Skyplane: Optimizing Transfer Cost and Throughput Using Cloud-Aware Overlays

Paras Jain, Sam Kumar, Sarah Wooders, Shishir G. Patil, Joseph E. Gonzalez, and Ion Stoica, *University of California, Berkeley*

https://www.usenix.org/conference/nsdi23/presentation/jain

This paper is included in the Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation.

April 17–19, 2023 • Boston, MA, USA

978-1-939133-33-5

Open access to the Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation is sponsored by

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Skyplane: Optimizing Transfer Cost and Throughput Using Cloud-Aware Overlays

Paras Jain, Sam Kumar, Sarah Wooders, Shishir G. Patil, Joseph E. Gonzalez, and Ion Stoica
University of California, Berkeley

Abstract

Cloud applications are increasingly distributing data across multiple regions and cloud providers. Unfortunately, wide-area bulk data transfers are often slow, bottlenecking applications. We demonstrate that it is possible to significantly improve inter-region cloud bulk transfer throughput by adapting network overlays to the cloud setting—that is, by routing data through indirect paths at the application layer. However, directly applying network overlays in this setting can result in unacceptable increases in cloud egress prices. We present Skyplane, a system for bulk data transfer between cloud object stores that uses cloud-aware network overlays to optimally navigate the trade-off between price and performance. Skyplane's planner uses mixed-integer linear programming to determine the optimal overlay path and resource allocation for data transfer, subject to user-provided constraints on price or performance. Skyplane outperforms public cloud transfer services by up to 4.6× for transfers within one cloud and by up to 5.0× across clouds.

1 Introduction

Increasingly, cloud applications transfer data across datacenter boundaries, both across multiple regions within a cloud provider (multi-region) and across multiple cloud providers (multi-cloud). This is in part due to privacy regulations, the availability of specialized hardware, and the desire to prevent vendor lock-in. In a recent survey [26], more than 86% of 727 respondents had adopted a multi-cloud strategy across diverse workloads. Thus, support for fast, cross-cloud bulk transfers is increasingly important.

Applications transfer data between datacenters for batch processing (e.g. ETL [9], Geo-Distributed Analytics [54]), and production serving (e.g. search indices [34]). Extensive prior work optimizes the throughput of bulk data transfers between datacenters within application-defined minimum performance constraints [34, 36, 38, 64]. All major clouds offer services for bulk transfers such as AWS DataSync [5], Azure AzCopy [22], and GCP Storage Transfer Service [31].

From the perspective of a cloud customer, transfer throughput and cost (price) are the two important metrics of transfers in the cloud. Thus we ask *how can we optimize network cost and throughput for cloud bulk transfers?* We study this question in the context of designing and implementing Skyplane, an open-source cloud object transfer system.

A seemingly natural approach is to optimize the routing protocols in cloud providers internal networks to support higher-throughput data transfers. Unfortunately, this is not feasible for two reasons. First, rearchitecting the IP layer routing protocol to optimize for high-throughput bulk transfer could be negatively impact other applications that are sensitive to network latency. Second, cloud providers lack a strong incentive to optimize data transfer to other clouds. Indeed, AWS DataSync [5], AzCopy [22], GCP Storage Transfer [31], AWS Snowball [62], and Azure Data Box Disk [12], all support data transfer *into*, but not out of, their respective clouds. Improvements to cross-cloud peering must be achieved with the cooperation of both the source and destination providers.

Skyplane's key observation is that we can instead identify *overlay paths*—paths that send data via intermediate regions—that are faster than the direct path. The throughput of the direct path from Azure's `Central Canada` region to GCP's `asia-northeast1` region is 6.2 Gbps. Instead, Skyplane can route the transfer via an intermediate stop at Azure's `US West 2` with a throughput of 12.4 Gbps for a 2.0× speedup (Fig. 1). Crucially, this can be implemented on top of the cloud providers' services without their explicit buy-in.

We are not the first to propose the use of overlay networks on the public Internet [8]. However, these techniques ignore two key considerations of public clouds: **price** and **elasticity**.

First, the highest-bandwidth overlay path may have an unacceptably high **price**. Cloud providers charge for data egress separately for each hop along the overlay path. To reduce the cost of the overlay, it is essential to transfer data along cheap paths to trade off price and performance. For example, in Fig. 1, one can achieve 13.9 Gbps by instead using Azure's `East Japan` region as the relay, but the cost would be 1.9× that of transferring data directly. In contrast, using Azure's

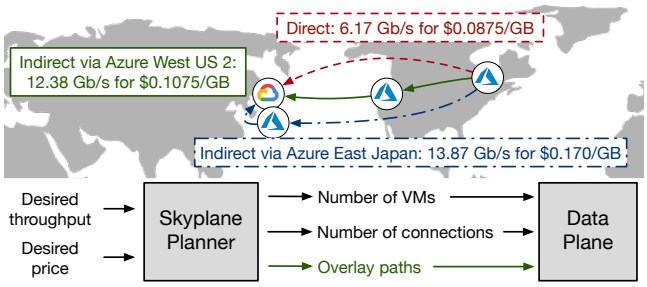


Figure 1: **Cloud-aware overlays:** Skyplane optimally transfers across cloud regions and providers subject to the user's cost and throughput requirements. Skyplane finds the visualized overlay path from Azure's `Central Canada` region to GCP's `asia-northeast1`, which is 2.0× faster but just 1.2× higher in price than the direct path.

`West US 2` region has only a 1.2× cost overhead with similar performance. Thus, Skyplane operates in a richer *problem space* than traditional application-level routing—one where cloud instance and cloud egress fees are significant.

Second, whereas the bandwidth between two nodes in a traditional network overlay [8] is considered "fixed," in Skyplane's setting it depends on **elasticity**—the ability to allocate more resources at each cloud region. For example, one can increase the capacity of any overlay path by simply allocating more VM instances in each cloud region. There are a limited number of physical machines at each cloud region, which cloud providers pass on to users in the form of instance limits. An overlay enables improved throughput beyond this limit. Thus, Skyplane operates in a richer *solution space* than traditional application-level routing—one where we must choose the number of VMs to use as relays due to cloud elasticity.

Skyplane addresses both **price** and **elasticity**, empowering users to navigate the trade-off between price and performance while leveraging the elasticity of cloud resources. Users can ask Skyplane to maximize bandwidth subject to a cost ceiling, or minimize cost subject to a bandwidth floor.

At the heart of Skyplane is a planner that computes a data transfer plan, subject to the user's constraints, that specifies the overlay path to use and amount of cloud resources to allocate along that path. Price and elasticity make it challenging to compute the plan. Our insight is that, with some care, planning can be formulated as *linear* constraints. Thus, Skyplane's planner can discover the optimal plan by solving a mixed-integer linear program (MILP), or closely approximate the optimal plan by solving a relaxed linear program (LP). Both can be accomplished using a fast, off-the-shelf solver.

Our Skyplane prototype[1] outperforms AWS DataSync by up to 4.6× and GCP Storage Transfer by up to 5.0×. Skyplane also outperforms academic baselines such as RON by 34% while reducing cost by 62%.

[1] https://github.com/skyplane-project/skyplane

2 Background

Network overlays In the early 2000s, network overlays emerged as a technique for *application-level routing* without the *participation of underlying network providers*. These network overlays can be designed to improve performance or reliability. Notable network overlays include Chord [60], Resilient Overlay Networks (RON) [8], Bullet [41], Baidu BDS [65] and Akamai's backbone [52, 58].

Although ISPs may have broad visibility into their networks, the metrics that ISPs use to select routes may not align with user preferences. Wide-area networks today do not allow specification of alternative routing preferences while network overlays provide applications a mechanism to control routing decisions. For example, Akamai uses a network overlay to reduce the latency of CDN misses while RON routes around network outages via an unaffected intermediate host.

RON is implemented by periodically measuring network performance via probes embedded in a fixed set of routers. When path outages occur, RON selects an intermediate relay router to circumvent the outage. This intermediate router is selected to have low packet loss or latency to/from the client and server. Optionally, RON can use a model of TCP Reno's throughput [53] to select intermediate routers. RON will generally select only a single intermediate node.

Wide-area networking in the cloud From the perspective of cloud customers, the cloud is *elastic*—additional resources can be allocated on demand. For example, an overloaded cloud application can leverage the cloud's elasticity by allocating additional VM instances. However, the physical reality of the cloud is that there are only finite resources at each region. Therefore, cloud providers impose *service limits* on their customers for resources such as VMs.

Each VM's network bandwidth is throttled according to its instance type. For example, an AWS `m5.8xlarge` instance can use at most 10 Gbps of network bandwidth, and an Azure `Standard_D32_v5` instance can use at most 16 Gbps of network bandwidth. Furthermore, only some of the available bandwidth can be used for egress traffic to another cloud provider. The policies differ by cloud provider. AWS limits VM egress bandwidth to the larger of 5 Gbps or 50% of total bandwidth [4], GCP limits VM egress bandwidth to any public IP address to 7 Gbps [30], and Microsoft Azure has no egress limit beyond the total bandwidth limit for a VM. Of course, the actual achievable TCP network bandwidth is subject to congestion control which may be less than the limit.

Cloud egress pricing Cloud providers charge egress prices for network traffic leaving a cloud region. Importantly, egress prices are assessed based on the *volume* of data transferred, not the rate at which it is transferred. Transferring a file at 10 Mbps or at 10 Gbps will result in the same egress charge. Egress charges introduce asymmetry in billing—there is no corresponding ingress charge for transfers into a cloud.

For intra-cloud transfers (i.e., transfers between two regions or zones in the same cloud), transfers between geographically distant endpoints are priced more than transfers between nearby endpoints. In contrast, inter-cloud transfers (i.e., transfers between two cloud providers) are billed at the same rate regardless of the transfer's geographic distance. For example, the egress price from a single Azure region is billed at the same rate for *any* destination outside of Azure, including any region in AWS or GCP [6, 29, 51].

Egress prices typically dominate the cost of a bulk transfers. For example, if a VM sends data at a rate of 1 Gbps for an hour on AWS with an Internet egress price of $0.09/GB, the total egress charge will total $40.50, which far exceeds the VM price of $1.50 (for `m5.8xlarge`) [6].

Cloud object storage AWS, Azure, and GCP provide object storage APIs that allow customers to save data attached to a string key. Data is stored immutably and therefore any updates require writing a new version. Unlike POSIX file systems, object stores do not provide atomic metadata operations (e.g., rename). Consistency models vary across providers. Cloud object stores store copies of a blob on multiple machines to improve availability and durability. Large objects support concurrent writes via sharding. Read throughput of a single shard may be limited by the provider (e.g. 60 MB/s for Azure [13]).

3 Overview of Skyplane

Skyplane allows applications to efficiently transfer large objects from an object store in one region to an object store in another cloud region or provider. To use Skyplane, the user installs the Skyplane client locally and configures it with access to cloud provider-supplied credentials. Then, the user submits a job, together with a constraint on price or bandwidth. The job specifies which objects to transfer, the source cloud provider and region, and the destination cloud provider and region. The constraint can have one of two forms: it can ask Skyplane to optimize either *bandwidth subject to a price ceiling*, or *price subject to a bandwidth floor*.

Skyplane itself comprises a *planner* (Fig. 1, bottom) and a *data plane* (Fig. 2). Given the user's job and constraint, the planner produces an optimal data transfer plan to complete the job subject to the constraint. The planner relies on a profile of the network throughput between different cloud regions. The data plane is responsible for executing the data transfer plan: allocating cloud resources (e.g., VMs), transferring data between them, and interacting with object stores.

3.1 Overlay formulation in Skyplane's planner

Suppose the user needs to transfer an object from a source cloud region, A, to a destination cloud region, B. A naïve object transfer system might spawn VMs in regions A and B, and transfer data via a TCP connection between the two VMs.

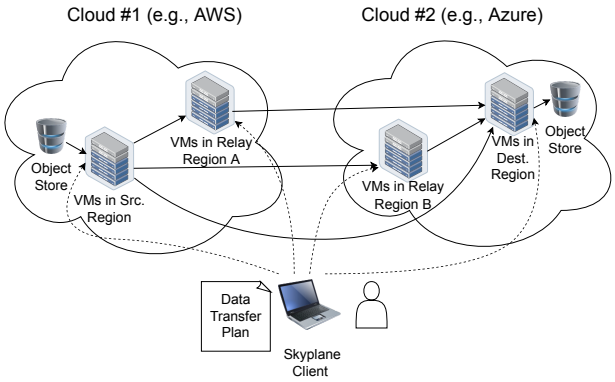


Figure 2: Skyplane splits an example data transfer over three paths: the direct path, and two indirect paths. Dashed lines indicate control orchestration (e.g., for spawning VMs) and solid lines depict the flow of object data.

Skyplane improves performance compared to this baseline by applying principles from overlay networks [8]. For example, Skyplane may identify a third cloud region, C, and transfer data from A to B via C. This is accomplished at the application layer; Skyplane will spawn a VM in region C, set up TCP connections from A to C and from C to B. We refer to intermediate regions like C as *relay regions*.

The baseline approach ($A \rightarrow B$) routes data along the "direct path," since it uses the default path provided by the Internet. However, Skyplane ($A \rightarrow C \rightarrow B$) routes data along the an "indirect path," that may not be on the Internet-provided default path. An indirect path may use multiple relays although a single relay is usually sufficient.

A key difference between Skyplane and classical overlay networks is that Skyplane takes price into account when choosing the overlay path to use for a job. Concretely, Skyplane's planner uses a *price grid* and a *throughput grid* to determine which indirect path to use. The price grid specifies the price of transferring data between each pair of cloud regions, in each direction. We computed the price grid based on information on the cloud providers' websites and from querying the cloud APIs. The throughput grid is collected by measuring the network, as we explain in the next subsection.

Note that throughput grid measurements are made using TCP connections, subject to TCP congestion control. Thus, the throughput grid measures the bandwidth available to *a single user* for transferring data, accounting for cross-traffic from other users' flows. We assume a high degree of statistical multiplexing in wide-area network traffic—in other words, that the bandwidth consumed by a single user's bulk transfer is negligible compared to the total available inter-region bandwidth. This allows a Skyplane user to compute a data transfer plan without regard to other users' bulk transfers using Skyplane or other bulk transfer tools—all cross traffic from other users is assumed to be accounted for in the throughput grid.

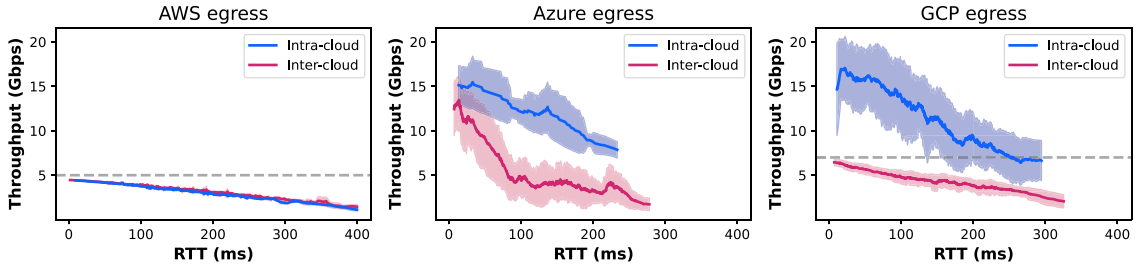


Figure 3: **Intra-cloud vs. inter-cloud links**: Inter-cloud links are consistently slower than intra-cloud links for network routes from Azure and GCP. Service limits are shown with a dashed line; GCP throttles inter-cloud egress to 7 Gbps while AWS throttles *all egress traffic* to 5 Gbps.

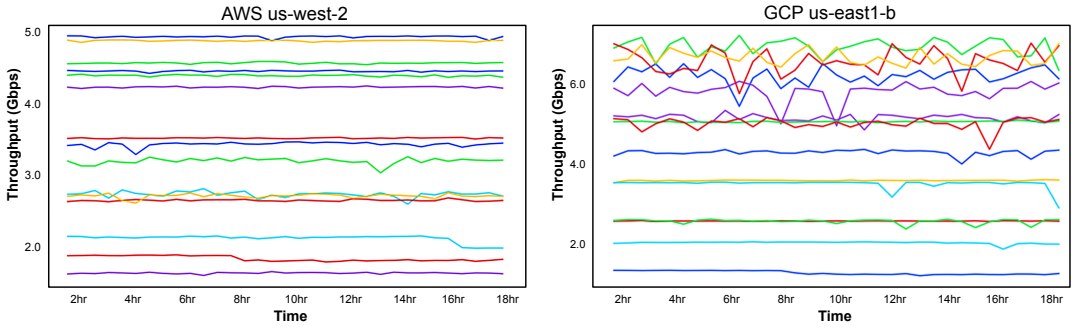


Figure 4: **Stability of egress flows over 18 hour period**: Continuous probes of cloud networks over one day reveal that routes from AWS have stable throughput over time. Paths between GCP regions are noisy but have a consistent mean.

As we show in the next subsection, the bandwidth of inter-region TCP connections is relatively stable in the short term, validating our assumption of high statistical multiplexing.

3.2 Profiling cloud networks

The planner relies on a profile of the network throughput between pairs of cloud regions. We collected a throughput grid by measuring the TCP goodput between each region pair using `iperf3`. In total, computing this profile cost approximately $4000 in egress charges.

Fig. 3 displays the relationship between network latency and throughput for profiling routes originating from GCP and Azure for our measured throughput grid. For GCP, we leverage internal IPs which improve intra-cloud bandwidth. For both GCP and Azure, intra-cloud routes had lower tail RTTs than inter-cloud routes. We observe that in both GCP and Azure, inter-cloud links are slower than intra-cloud links. As Azure has no service limit for egress bandwidth, we see the fastest intra-cloud links achieve up to the NIC capacity of 16 Gbps. However, both GCP and AWS encounter egress throttling at 7 Gbps and 5 Gbps respectively.

A natural question is how frequently the throughput grid must be re-measured. Fig. 4 visualizes achieved throughput from AWS `us-west-2` and GCP `us-east1-b` taken every 30 minutes over an 18 hour timespan. Throughput is very stable over time for both inter-cloud and intra-cloud routes from AWS `us-west-2`. Routes from GCP `us-east1-b` to AWS destinations is similarly very stable but intra-cloud routes to GCP destinations are less stable. Regardless, the overall rank order of regions by throughput remains mostly consistent over medium-term timescales. Thus, it should be sufficient to profile networks relatively infrequently (i.e. every few days). In practice, this information could be collected by third-party service, or measured via active probing along live transfers.

3.3 Skyplane's data plane

Skyplane's data plane executes data transfers using the plan computed by Skyplane's planner. Ephemeral VMs for a single transfer, called "gateways," are provisioned in the source region, destination region, and overlay regions for a transfer plan. Each source gateway reads a small shard of data from the object store and transfers data via intermediate gateways to the destination where the shard is written.

Skyplane reads data from an object store in the source cloud region and writes data to an object store in the destination cloud region. We focus on the object stores provided as a service by AWS S3, Azure Blob Storage, and Google Storage. Unlike a traditional overlay network, there is no central Skyplane service that allocates resources to each user from a pool of "Skyplane resources." Instead, Skyplane can be understood

as a local service run by each user that is invoked when an application needs to transfer data. Skyplane directly allocates cloud resources on the user's behalf when processing a job, and manages those resources to transfer the user's data across cloud regions. This allows Skyplane to allocate and manage each user's resources according to their cost and performance objectives, independently from the cloud providers' existing data transfer services, while relying on clouds to offer a large pool of resources and manage isolation between users.

4 Principles of Skyplane's planner

Skyplane's planner[2] is responsible for developing a plan for transferring data across the wide area to complete an object transfer job submitted by a user or their application (Fig. 5). This plan describes the overlay path and the amount of cloud resources to allocate along that path to facilitate the transfer.

Skyplane's planner supports two modes:

Cost minimizing: The planner will minimize cost subject to an application-specified throughput constraint.

Throughput maximizing: The planner will maximize throughput subject to an application-specified cost constraint.

As we will describe in §5, Skyplane finds the optimal plan by formulating it as an Mixed-Integer Linear Program (MILP) and using a fast but exponential-time solver. This section describes the degrees of freedom available to the optimizer to navigate the price-performance trade-off for the user's specified constraint. Our goal is to describe what aspects of the plan are at the planner's disposal, justify why it is reasonable to vary those aspects of the plan, and describe certain techniques available to the planner to manage the price-performance trade-off. Note that the planner is not directly programmed to use these techniques; they are merely patterns that it discovers in the course of finding the optimal MILP solution.

4.1 Achieving low instance and egress costs

That bandwidth costs dominate the cost of data transfer (§2) is both a challenge and an opportunity for Skyplane. It is an opportunity because it allows Skyplane to be competitive with the price of using data transfer tools provided directly by the cloud providers (e.g. AWS DataSync, AzCopy, GCP Cloud Transfer Service), as those tools incur bandwidth costs but not instance costs. It is a challenge for Skyplane because it implies that, used naïvely, indirect paths are much more expensive than direct paths. This is because egress bandwidth is charged for each hop along the path. For example, for a path $A \to C \to B$, the bandwidth cost must be paid for both $A \to C$ and $C \to B$, which could be *double* the cost of transferring over the direct path. As a result, it is crucial for Skyplane's optimizer carefully manage egress transfer costs.

[2]Explore Skyplane's planner at `https://optimizer.skyplane.org`

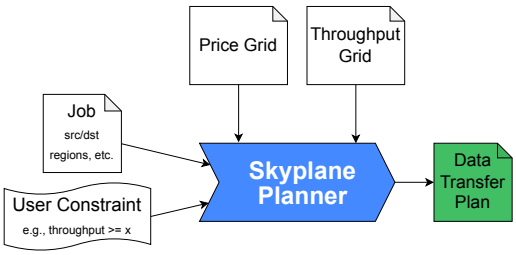


Figure 5: Skyplane's planner considers throughput and cost constraints from the user along with per-cloud price information and an inter-region throughput profile grid to determine the optimal data transfer plan.

4.1.1 Choosing the relay region

One way for Skyplane to manage the additional cost associated with indirect paths is to carefully choose the relay region C to minimize this cost. For example, suppose that a user needs to transfer an object from AWS `us-west-2` (region A) to Azure `UK South` (region B). The direct path $A \to B$ would require the user to pay $0.09 per GB, the cost of bandwidth leaving AWS' network. If the relay region C is chosen in `us-central-1` or `us-east-1`, then the overall bandwidth price will only increase slightly; while the $C \to B$ transfer still incurs $0.09 per GB, as data is leaving AWS' network, the $A \to C$ bandwidth only costs $0.02 per GB, as it is an intra-continental transfer within the cloud provider's network. Skyplane's planner can use the throughput and price grids to identify relay regions that improve the performance of the transfer while minimizing additional bandwidth costs.

4.1.2 Combining multiple paths

Another way to manage the cost of indirect paths is to split the data transfer over multiple paths, in order to make fine-grained trade-offs between price and performance. For example, suppose that Skyplane identifies a high-bandwidth indirect path, but that the path is more expensive than the user's price ceiling. Skyplane can still benefit partially from that indirect path by sending part of the data over that path, at higher cost, and the remaining data over the direct path $A \to B$, at lower cost. Thus, Skyplane may average the price and performance of multiple paths, when doing so allows Skyplane to more optimally satisfy the user's constraints.

4.2 Parallel TCP for high bandwidth

Skyplane uses parallel TCP connections—that is, bundles of TCP connections—to achieve high goodput over a chosen path. This is a well-known technique for achieving good performance, particularly for wide-area transfers [1, 59]. Our Skyplane implementation uses up to 64 outgoing connections for each VM instance, as we empirically measured that using

additional connections typically resulted in diminishing benefits in aggregate goodput. When collecting measurements for the throughput grid, we make sure to use 64 parallel connections to measure the achievable TCP goodput for each ordered pair of regions.

It is known that using multiple TCP streams in parallel may cause an application to obtain more than its "fair share" of bandwidth [25, §A.1], particularly in contexts where networks are running at nearly 100% utilization [36]. Our view is that, despite this, it is acceptable to use multiple TCP connections in parallel in the context of Skyplane. There are three reasons for this. First, it is common for applications to use parallel TCP, including for workloads like bulk data transfer [1, 44]. It is important for Skyplane to appropriately compete with such applications for limited bandwidth. Second, the user pays the cloud provider for bandwidth, both in the form of the bandwidth price (total amount transferred) and the instance price (rate at which data can be transferred), and it is natural for users to be able to make use of the bandwidth that they pay for. Third, cloud providers control the datacenter network, and can shape traffic in the presence of congestion to ensure that each customer gets a fair share of bandwidth.

4.3 Multiple VMs for high bandwidth

For a given overlay path, Skyplane must allocate sufficient resources along the path to achieve high bandwidth. However, the achievable outgoing bandwidth from a VM instance is limited, as described in §2.

Therefore, Skyplane may allocate multiple VM instances at certain regions along the path, to increase *aggregate* data transfer rate of the VMs at each region. Although simply using larger VMs may seem like a viable alternative, it is less effective than using multiple instances due to per-instance bandwidth limits. Skyplane uses a fixed VM size, and its planner chooses how many instances to allocate in each region, under the assumption that TCP goodput scales linearly with the number of allocated VM sizes.

It may seem that Skyplane can achieve an arbitrarily high bandwidth by spawning many instances in each region. Unfortunately, this simple strategy does not work because cloud resources are not perfectly elastic. The finite capacity for VMs in a datacenter is passed down to cloud customers in the form of service limits, which limit the number of VM instances, and therefore the amount of network bandwidth, that users can allocate in each region. While users can request limit increases, these are ultimately subject to resource availability. To model this, Skyplane's planner takes into account a limit on the number of instances that a user can allocate per region.

5 Finding optimal transfer plans

Skyplane's planner searches for cost-efficient high-throughput transfer plans that jointly specify the overlay path, TCP connections between regions and VMs to provision per region.

Variables	
$\mathrm{F} \in \mathbb{R}_+^{\lvert V\rvert\times\lvert V\rvert}$	*Throughput grid*
$\mathrm{N} \in \mathbb{Z}_+^{\lvert V\rvert}$	*VMs per region*
$\mathrm{M} \in \mathbb{Z}_+^{\lvert V\rvert\times\lvert V\rvert}$	*TCP conn. per region*
Constraint: goal throughput	
$\textsc{Tput Goal} \in \mathbb{R}_+^{\lvert V\rvert\times\lvert V\rvert}$	*User's desired throughput*
Constants: provider limit	
$\textsc{Limit}^{link} \in \mathbb{R}_+^{\lvert V\rvert\times\lvert V\rvert}$	*Throughput grid limit*
$\textsc{Limit}^{conn} \in \mathbb{Z}_+^{\lvert V\rvert\times\lvert V\rvert}$	*TCP connection limit*
$\textsc{Limit}^{ingress} \in \mathbb{Z}_+^{\lvert V\rvert}$	*VM limit*
$\textsc{Limit}^{egress} \in \mathbb{Z}_+^{\lvert V\rvert}$	*Egress bandwidth limit*
Constants: provider cost	
$\textsc{Cost}^{egress} \in \mathbb{R}_+^{\lvert V\rvert}$	*Egress cost ($/Gbit)*
$\textsc{Cost}^{VM} \in \mathbb{R}_+^{\lvert V\rvert}$	*VM cost ($/s)*

Table 1: Symbol table for Skyplane's ILP formulation.

At the core of Skyplane's planner is an optimizer that finds the optimal plan using off-the-shelf Linear Programming (LP) solvers. We formalize the constraints of our problem as Mixed Integer LP (MILP) which can quickly be solved in under 5 seconds with an open-source solver. The problem can be further relaxed into a continuous LP which is solvable in worst-case polynomial time via interior point methods [39].

Independently optimizing for each variable then combining partial solutions would not guarantee a globally optimal solution. It is therefore important that Skyplane's planner models all variables in an integrated search space to obtain provably optimal data transfer plans.

5.1 Cost minimizing overlay paths

Flow networks can naturally represent overlay networking topologies like those used by Akamai [58]. We start with a min-cost flow problem. The following primal LP finds the optimal flow matrix $\mathrm{F} \in \mathbb{R}_+^{\lvert V\rvert\times\lvert V\rvert}$ for a network topology graph $G = (V, E)$ where nodes represent regions and edges are links:

$$\begin{aligned} \underset{\mathrm{F}}{\text{arg min}} \quad & \langle C, \mathrm{F}\rangle \\ \text{subject to} \quad & \textstyle\sum_{(c,v)\in E} \mathrm{F}_{c,v} \geq \textsc{Tput Goal} \\ & \textstyle\sum_{(u,v)\in E} \mathrm{F}_{u,v} = \sum_{v,w} \mathrm{F}_{v,w} \quad \forall v \in V - \{s,t\} \\ & 0 \leq \mathrm{F} \leq \textsc{Limit}^{link} \end{aligned} \tag{1}$$

where s and t are the source and destination regions, $\textsc{Limit}^{link} \in \mathbb{R}_+^{\lvert V\rvert\times\lvert V\rvert}$ is the maximum capacity for each link and $C \in \mathbb{R}_+^{\lvert V\rvert\times\lvert V\rvert}$ is the cost per unit of bandwidth between regions. We use the same notation for matrix and vector inner products: $\langle C, \mathrm{F}\rangle = \sum_{u,v} C_{u,v}\mathrm{F}_{u,v}$.

5.1.1 Objective: Minimize cost from egress and VMs

Min-cost flows do not accurately reflect the cost of transfers in the cloud. The total cost of a transfer in Skyplane includes *egress cost* and *VM cost*. Note that this objective is not linear; we present a linear reformulation in Sec. 5.1.1. We present the full objective is in the in Equation 4a.

Modeling egress cost Unlike physical networks, virtual networks in the cloud will charge the same amount if 1GB of data is sent at 1 Mbps or 10 Gbps. Transfers are priced according to *egress volume* ($ per GB, COST^{egress}) rather than *bandwidth* ($ per Gbps). We can update the cost function to instead model the transfer cost by first computing how much the overlay path costs to run per unit time and then scale that by the runtime for a transfer. We denote the total volume of the transfer as VOLUME. Total egress cost is then:

$$\underbrace{\langle \mathrm{F}, \text{COST}^{egress} \rangle}_{\text{Egress cost per s}} * \underbrace{\text{VOLUME} \div \textstyle\sum_{v \in V} \mathrm{F}_{s,v}}_{\text{Transfer time}} \tag{2}$$

Modeling VM cost Multiple VMs can increase aggregate bandwidth as discussed in Sec. 4.3. To optimally trade-off parallel VMs with the overlay, we introduce a new decision variable $\mathrm{N} \in \mathbb{Z}_+^{|V|}$ that models the number of instances use to transfer data per region. VM count per region may vary due to asymmetric egress and ingress limits. To accurately consider transfer costs from VMs, we add the the following instance cost expression to Equation 2 where COST^{VM} is a vector containing the cost per second per VM in each region:

$$\underbrace{\langle \mathrm{N}, \text{COST}^{VM} \rangle}_{\text{VM cost per s}} * \underbrace{\text{VOLUME} \div \textstyle\sum_{v \in V} \mathrm{F}_{s,v}}_{\text{Transfer time}} \tag{3}$$

Linear reformulation of the objective As written, the objective in Equation 4a is not linear due to a product of variables between F and N. By reformulating the problem to instead consider finding a plan that provides *exactly* TPUT GOAL (instead at least), the runtime for the transfer can be reduced to a constant VOLUME ÷ TPUT GOAL.

5.1.2 Constraints: Cloud provider service limits

Resources are not infinite at cloud regions; providers limit the number of VMs that a user may provision and in some cases, providers may throttle the performance of ingress and egress.

Per VM ingress and egress limits AWS and GCP each throttle egress from their clouds via SDN policies. For AWS, instances with 32 cores or less are limited to 5 Gbps. For GCP, individual flows are limited to 3 Gbps and total egress is service limited to 7 Gbps. Ingress is bottlenecked by VM NIC bandwidth. We constrain the maximum ingress bandwidth per VM to $\text{LIMIT}^{ingress}$ via Constraint 4f and the maximum egress bandwidth per VM to LIMIT^{egress} via Constraint 4g.

Constraining TCP connections Using parallel TCP connections is a well known approach to improve WAN performance as discussed in Section 4.2. Yet, bandwidth does not scale linearly with connections (Figure 9a). We introduce a decision variable $\mathrm{M} \in \mathbb{Z}_+^{|V| \times |V|}$ representing the number of connections between a *pair of regions* (not per VM pair). Constraint 4b ensures M is constrained by N and LIMIT^{conn} (typically 64 per VM). We then limit the total incoming and outgoing connections with Constraints 4i and 4h.

Per-region VM limits We introduce the variable $\mathrm{N} \in \mathbb{Z}_+^{|V|}$ to denote the number of VMs per region. N must be under the global instance cap in Constraint 4j. The optimizer linearly scales the maximum number of egress TCP connections per region by the number of VMs provisioned in each region.

5.1.3 Continuous relaxation of MILP

To improve solve times, N and M are relaxed into real valued variables $\mathrm{N} \in \mathbb{R}_+^{|V|}$ and $\mathrm{M} \in \mathbb{R}_+^{|V| \times |V|}$. Rounding variables down performs comparably to randomized rounding with solutions $\leq 1\%$ from optimal. The relaxed problem has worst case polynomial time complexity [39].

5.1.4 Full formulation of the cost optimal solver

All variables and constants are listed in Table 1. The full formulation of Skyplane's optimizer is:

$$\underset{\substack{\mathrm{F},\,\mathrm{N}\\ \mathrm{M}}}{\arg\min} \quad \frac{\text{VOLUME}}{\text{TPUT GOAL}} \left(\langle \mathrm{F}, \text{COST}^{egress} \rangle + \langle \mathrm{N}, \text{COST}^{VM} \rangle \right) \tag{4a}$$

subject to

$$\mathrm{F} \leq (\text{LIMIT}^{link} \odot \mathrm{M}) \div \text{LIMIT}^{conn} \tag{4b}$$

$$\textstyle\sum_{v \in V} \mathrm{F}_{s,v} \geq \text{TPUT GOAL} \tag{4c}$$

$$\textstyle\sum_{u \in V} \mathrm{F}_{u,t} \geq \text{TPUT GOAL} \tag{4d}$$

$$\textstyle\sum_{u \in V} \mathrm{F}_{u,v} = \sum_{u \in V} \mathrm{F}_{v,u} \quad \forall v \in V - \{\mathrm{s}, \mathrm{t}\} \tag{4e}$$

$$\textstyle\sum_{u \in V} \mathrm{F}_{u,v} \leq \text{LIMIT}_v^{ingress} * \mathrm{N}_v \quad \forall v \in V \tag{4f}$$

$$\textstyle\sum_{v \in V} \mathrm{F}_{u,v} \leq \text{LIMIT}_u^{egress} * \mathrm{N}_u \quad \forall u \in V \tag{4g}$$

$$\textstyle\sum_{v \in V} \mathrm{M}_{u,v} \leq \text{LIMIT}^{conn} * \mathrm{N}_v \quad \forall u \in V \tag{4h}$$

$$\textstyle\sum_{u \in V} \mathrm{M}_{u,v} \leq \text{LIMIT}^{conn} * \mathrm{N}_u \quad \forall v \in V \tag{4i}$$

$$N_v \leq \text{LIMIT}^{VM} \quad \forall v \in V \tag{4j}$$

5.2 Throughput maximizing overlay paths

Directly solving for a throughput maximizing path under a cost ceiling is non-trivial as we cannot use the linear reformulation of the cost objective. We can approximate a solution by solving for the minimum cost transfer plan at a range of many throughput goals. The result of this procedure is a Pareto

frontier curve (as shown in Fig. 9c). A throughput maximizing solution can be extracted from this curve. The quality of approximate solution will depend on how many samples are used. A single AWS `c5.9xlarge` instance can evaluate 100 samples in under 20 seconds.

6 Implementation of Skyplane

We implemented Skyplane in Python 3. Skyplane's planner uses the proprietary Gurobi library to solve MILP instances (used in our evaluation), but the Coin-OR library can be used instead to avoid this dependency. Our implementation currently supports the three major cloud providers: Amazon Web Services, Microsoft Azure, and Google Cloud Platform.

We use `m5.8xlarge` instances on AWS, as smaller VM sizes were subject to burstable networking performance, which we wished to avoid [4, 7]. For consistency, we used `Standard_D32_v5` instances on Microsoft Azure and `n2-standard-32` instances on Google Cloud.

A user initiates a transfer from their application with the *Skyplane client*. The client provisions VMs in each region according to the transfer plan and runs the *Skyplane gateway* program on each VM. The gateway is responsible for actually reading from source object stores, relaying data through overlay regions and writing to destination object stores.

While transfer time is dominated by network throughput, the time to spawn gateway VMs contributes to the transfer latency. To minimize unnecessary bloat in VM images, we use compact OSes such as Bottlerocket [3] and package dependencies via Docker.

Skyplane assumes that objects are broken up into small *chunks* of approximately equal size. Applications can often do this without significant burden; for example, machine learning applications store data as `TFRecords`, which are easy to split into small chunks. This allows Skyplane to read and write data quickly from and to cloud object stores, by issuing many read/write operations in parallel to different chunks.

To mitigate the impact of straggler connections, Skyplane dynamically partitions data across TCP connections as they become ready to accept more data. This is in contrast to tools like GridFTP [1], which assign data blocks to connections in a round-robin fashion. The downside is that, for plans that use multiple overlay paths, the amount of data sent on each path may deviate from the targets computed at planning time, which could cause the actual cost of transferring data to deviate from the cost predicted by Skyplane's planner.

To avoid overflowing buffers at relay regions, Skyplane uses hop-by-hop flow control to stop reading data from incoming TCP connections when a VM's queue of chunks reaches capacity. Bufferbloat-type problems [28] are not a concern for Skyplane, with regard to queued chunks, as we pipeline transfers to optimize for throughput instead of latency.

7 Evaluation

To evaluate Skyplane, we investigate transfer time and price. We will sometimes use transfer throughput as a proxy for transfer time. In our price calculations, we include both instance cost and egress cost.

7.1 Experimental setup

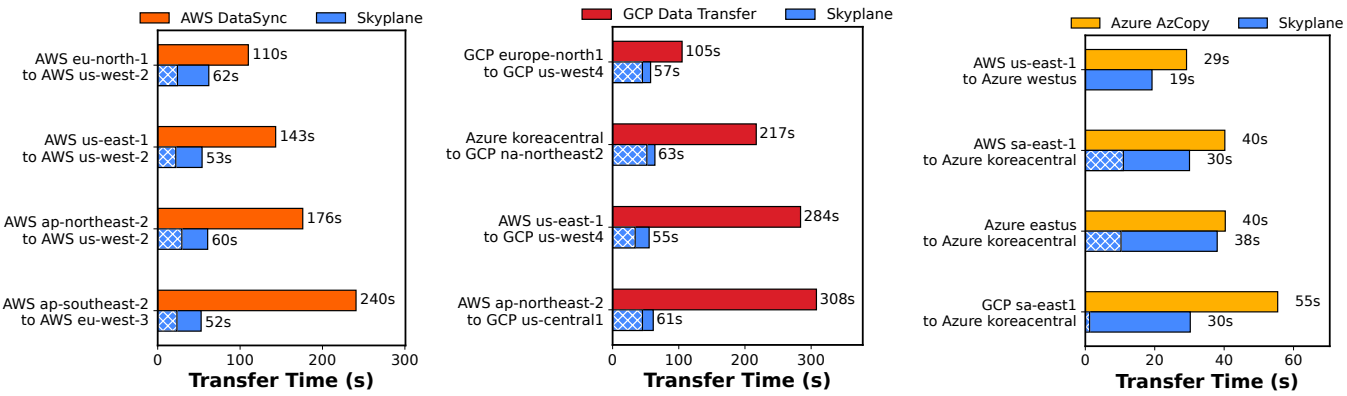
We evaluate Skyplane with 20 AWS regions, 24 Azure regions and 27 GCP regions. For all experiments, we use public IP addresses attached to the VMs for transferring data. In some cases, one can achieve better performance for intra-cloud overlay hops by using private IP addresses assigned to each VM. For GCP this yields higher performance; for AWS and Azure it may yield higher performance, but requires peering virtual networks which incurs additional fees.

Furthermore, Azure and GCP allow one to select *network tiers* to control whether data is transferred via the cloud provider's network or via the public Internet. The Skyplane prototype utilizes external IPs over standard network tiers. That said, Skyplane is not incompatible with optimizations like VPC peering or hot-potato routing tiers to reduce cost and improve performance which we leave to future work. We use the CUBIC congestion control protocol in experiments.

7.2 How much faster is Skyplane than existing data transfer solutions?

Existing cloud providers offer data transfer tools such as AWS DataSync, GCP Storage Transfer, and Azure AzCopy for low-cost transfers of bulk data into their respective clouds. These tools do not disclose what mechanisms they use to transfer data—for example, the number of VMs and TCP connections (if any) used for a transfer, or the QoS (if any) associated with the network traffic. When evaluating Skyplane, we restrict Skyplane to use at most 8 VMs in each region. This is conservative; for example, on equalizing $/GB for some routes, Skyplane could provision *up to 262 VMs* per region within DataSync's service fee. Moreover, while these services only support data transfer *into* their respective clouds, Skyplane supports data transfer between every region pair.

We consider transferring the training and validation set for ImageNet [23]. We specifically use the `TFRecords` as generated by Google as part of the Cloud TPU benchmark example [23]. We evaluate flows between regions within a single cloud (intra-provider) and between clouds (inter-provider). We expected that data transfer within each cloud provider (e.g., between AWS's `us-east-1` and AWS's `us-west-1`) to perform well as they have full visibility into their networks and can utilize private interfaces with higher performance than over public API. For example, Azure Blob Storage throttles per-object reads for third-party VMs [50]. Our experiments



(a) AWS DataSync comparison

(b) GCP Storage Transfer comparison

(c) Azure AzCopy comparison

Figure 6: **Comparison to cloud transfer systems**: The thatch pattern in each bar represents the storage I/O overhead.

did observe this behavior. However, Skyplane benefits from parallelizing the transfers.

We compare against AWS DataSync, GCP Storage Transfer and Azure AzCopy in Fig. 6. We evaluated Skyplane with a cost budget cap that is lower than the service fee for cloud transfer services in all our experiments. For each source-destination pair, we additionally measured the time to transfer procedurally-generated data using Skyplane; this allows us to break out the overhead of reading and writing to cloud storage as a "thatched" region in each bar. Skyplane significantly outperforms AWS DataSync and GCP Cloud Transfer in all configurations. In certain cases, Azure AzCopy performs about as well as Skyplane. We chose the `koreacentral` region because we expected the greatest improvements from the overlay in that region; however, storage overheads (the "thatched" regions of the bars), not networking overheads, dominated the runtime. It is possible that AzCopy avoids the Azure Blob Storage I/O overhead that dominates Skyplane's transfer time by leveraging Azure's `Copy Blob From URL` API call to download data directly into the servers running Azure Blob Storage [11].

7.3 How much faster are the overlay paths?

The planner optimally explores the trade-off between improved throughput and cost for cloud data transfers. We explore solving for the optimal transfer path between all pairs of clouds regions between all cloud providers. We evaluated 22 AWS regions, 23 unrestricted Azure regions and 27 GCP regions which leads to 5,184 possible replication routes. It would be too expensive to transfer a large amount of data along each path in order to measure the empirical achieved throughput; therefore we use the planner to generate a plan and compare the resulting plan with the direct path, both in terms of expected throughput and cost. We compute predicted costs for transferring a 50 GB dataset between each possible source and destination. We report the speedup relative to Skyplane with a direct connection between each set of instances. Notice that the baseline is itself an ablation of Skyplane and it generally outperforms existing cloud transfer services to begin with (see §7.2).

The results are shown in Fig. 7. For each pair of source and destination clouds, we show distribution of predicted throughputs across region pairs, both with Skyplane's planner restricted to the direct path and allowing Skyplane's planner to use overlay paths. The results show that Skyplane's overlay routing meaningfully improves achievable throughput between cloud regions. Note that transfers out of AWS cannot exceed 5 Gbps and transfers leaving GCP cannot exceed 7 Gbps due to these cloud providers' caps on egress bandwidth.

7.4 Where are transfer bottlenecks?

To understand how the overlay improves throughput, we characterize the fraction of transfers that are bottlenecked at each location. In Fig. 8, we visualize the percentage of transfers from §7.3 that were bottlecked at a VM in the source region, the network link leaving the source region, a VMs in optional overlay regions, a network links leaving an overlay region, and a VM in the destination region. We consider a particular location to be a bottleneck if utilization is over 99%. Multiple locations may simultaneously be a bottleneck for one transfer.

For Skyplane with overlay routing disabled, the network link from the source to the destination region is the most common bottleneck for transfers. In a small set of cases, the source VM is a bottleneck for the transfer. Generally, the direct path is not fast enough to saturate the maximum egress bandwidth limit for a VM. The overlay shifts source link bottlenecks by reducing the number of transfers bottlenecked by the source link by 32%. The bottleneck shifts to the source VM or in some cases a network link leaving an overlay region.

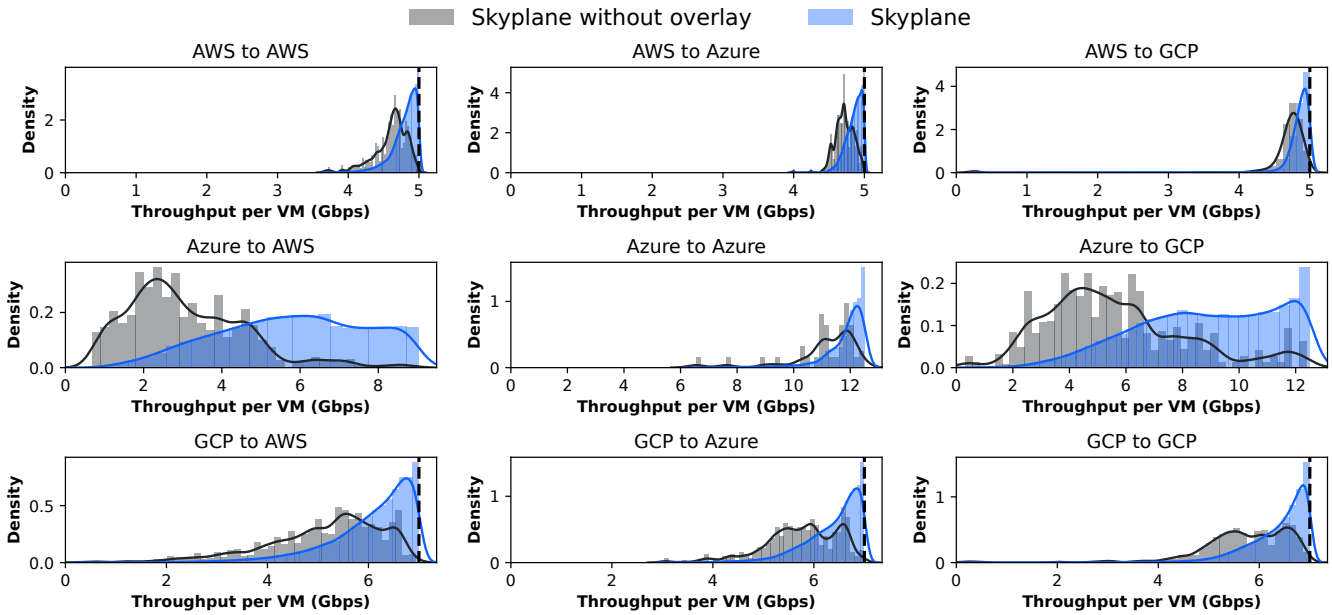


Figure 7: **Ablation of predicted overlays**: Overlay routes improve throughput per VM instance. We visualize the distribution of predicted throughput by the planner with all optimizations enabled (Skyplane) and with all optimizations except for overlay routing (Skyplane without overlay). The AWS and GCP egress limits are displayed with a dashed line.

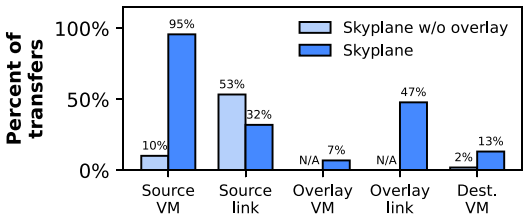


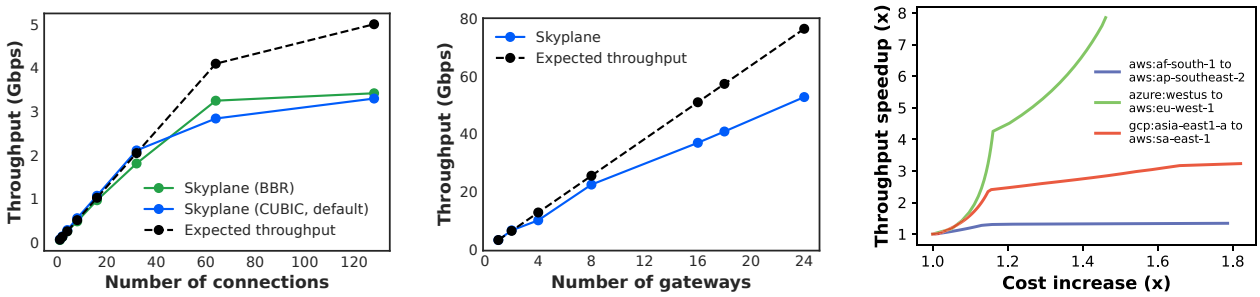
Figure 8: **Transfers bottlenecked at each location**: For transfers in Fig. 7, we visualize what percentage of transfers were bottlenecked at various locations. Enabling the overlay shifts bottlenecks from the network to the VM.

7.5 Skyplane microbenchmarks

Impact of parallel TCP connections Fig. 9a shows the impact of varying the number of parallel TCP connections used to transfer data between VMs. For this experiment, the source VM was located in AWS `ap-northeast-1` and the destination VM was located in AWS `eu-central-1`. Skyplane transfers 32 GB of synthetic, procedurally-generated data in these experiments to avoid incurring object store I/O overheads and thereby isolate network performance. The black dashed line shows the expected throughput, assuming that bandwidth scales linearly with the number of parallel TCP connections up to AWS' 5 Gbps egress cap. The blue line shows Skyplane's achieved throughput, and the green line uses Skyplane's achieved throughput using the BBR congestion control algorithm (used only this experiment). For this experiment, the source VM was located in AWS `ap-northeast-1` and the destination VM was located in AWS `eu-central-1`. Skyplane's achieved throughput plateaus below the 5 Gbps egress cap, and 64 connections is enough to come close.

Impact of parallel VMs Fig. 9b shows the impact of using multiple VMs in each region to achieve higher aggregate throughput. The black dashed line shows the expected throughput, assuming that bandwidth scales linearly with the number of VMs. Although Skyplane's performance is significantly less than the expected throughput for a large number of gateways, the graph shows that using parallel VMs is an effective way for Skyplane to scale its aggregate bandwidth. Additionally, using parallel VMs is a particularly valuable tool in the context of inter-cloud transfers, as Skyplane can use multiple VMs in one cloud provider to circumvent the egress limit. For example, for an overlay hop from an AWS region to an Azure region, one may allocate many instances in AWS but few in Azure, to account for AWS' egress cap.

Trade-off between cost and throughput Fig. 9c shows the impact on overlay path throughput as the price budget is varied. We adjusted the cost budget afforded to the planner (x-axis), and plot the throughput predicted by the planner for the output plan (y-axis). We show three routes where the overlay benefits are considerable (Azure `westus` to AWS `eu-west-1`), good (GCP `asia-east1-a` to AWS `sa-east-1`) and minimal (AWS `af-south-1` to AWS



(a) TCP connections versus throughput (b) Number of gateway VMs versus throughput (c) Predicted planner throughput versus cost

Figure 9: **Skyplane ablations**: We evaluate the impact of parallel TCP connections, parallel gateway VMs and overlay cost.

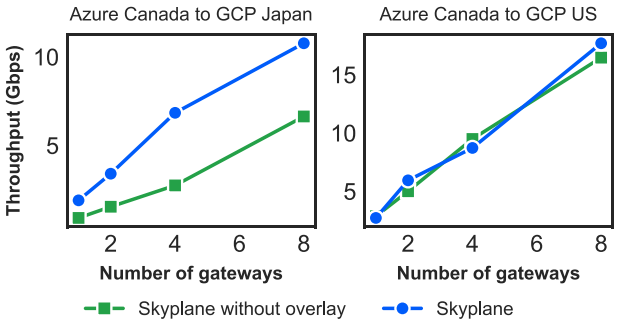


Figure 10: **Scaling VMs versus overlay**: In situations where the direct path is slow, the overlay is faster than simply scaling the number of VMs used alone.

`ap-southeast-2`). As the cost budget increases, Skyplane uses increasingly complex overlay topologies, adding new overlay paths as the instance limit (1 VM, in this case) is saturated in each region. Each elbow in the plot (e.g. 1.2× for the Azure to AWS route) represents a point where Skyplane adds a new overlay route via a faster but more costly region. At some point, the planner cannot increase throughput further as the overlay network is saturated.

Is it better to use VMs to form overlay paths or parallelize the direct path? Given a limited number of VMs (§4.3), a natural question is whether it is better to use those VMs to form overlay paths or to parallelize the direct path. In Fig. 10, we evaluate Skyplane with and without the overlay enabled for various numbers of VMs in the context of an inter-continental transfer and an intra-continental transfer. For the inter-continental transfer, using the VMs with overlays enabled provides a 2.08× geomean speedup compared to using those VMs to parallelize the direct path. However, for the intra-continental transfer, there is little benefit to using VMs in overlay paths (1.03× geomean speedup).

Table 2: **Comparison with academic baselines**: Skyplane outperforms RON's path selection heuristic implemented in Skyplane [8].

Method	Time	Throughput	Cost
GCT GridFTP [1, 10] (1 VM)	133s	1.03 Gbps	$1.40
Skyplane (1 VM, direct)	**73s**	**1.71 Gbps**	$1.40
Skyplane w/ RON routes (4 VMs) [8]	21s	6.02 Gbps	$2.27
Skyplane (cost optimized, 4 VMs)	32s	3.88 Gbps	$1.56
Skyplane (throughput optimized, 4 VMs)	**16s**	**8.07 Gbps**	$1.59

7.6 Comparison against academic baselines

In Table 2, we compare Skyplane with RON [8] and the community-maintained fork [10] of GridFTP [1] for a 16 GB data transfer from Azure `East US` to AWS `ap-northeast-1`. To isolate network throughput from I/O overheads, we benchmark the transfers without object stores (VM to VM only).

We use the open-source GCT fork of GridFTP [10]. Although GCT GridFTP theoretically supports striped transfers across multiple machines, we were unable to find a supported non-commercial implementation. To make a fair comparison, we run both GCT GridFTP and Skyplane with a single VM per region. Skyplane is 1.6× faster than GCT GridFTP.

We implement RON's path selection heuristic in Skyplane to compare overlays between RON and Skyplane. Our results show that Skyplane has better cost and throughput than RON. Skyplane with routes from RON's path selection heuristic achieves 3.5× higher throughput than Skyplane with a single VM but at 62% cost overhead. Skyplane's planner instead finds overlay paths with up to 4.7× higher throughput than the direct path within a 14% cost overhead.

8 Related Work

Skyplane builds on the overlay network literature [8, 16, 58]. As discussed in §1, Skyplane adapts classical overlays to the cloud setting, accounting for the price of network bandwidth

and leveraging the elasticity of cloud resources. CRONets [16] briefly discusses cost, but focuses on comparing cloud-based options to private leased lines. Unlike Skyplane, it does not discuss how to manage the cost of cloud resources. Lai et al. [46] find relay regions improve throughput in AWS when utilizing a single TCP connection but find the 2 Gbps instance NIC limit from their chosen instance class limits the benefit of overlay paths. CloudCast [56] examines the use of triangle overlays in the cloud to reduce network latency while Skyplane examines throughput.

Several existing efforts [27, 49, 55] aim to optimize bulk data transfers by reducing the amount of data transferred. Such techniques are complementary to Skyplane; one can first apply these techniques to reduce the amount of data to transfer, and then apply Skyplane's techniques to transfer that reduced data efficiently. Unlike Skyplane, these works do not use cost when selecting the network path to use for a transfer.

Another line of research aims to improve bulk data transfers by improving resource management. GridFTP [1] is a tool for wide-area transfers that techniques such as using multiple machines and TCP connections. GridFTP sends all data over the direct path and does not utilize overlays. Khanna et al. [40] explore application of network overlays to GridFTP but do not consider elasticity and egress price in the cloud. Other solutions, like PSockets [59], also use parallel TCP connections for high bandwidth. Pied Piper [14] also explored how cloud resource elasticity could be used to improve cloud data transfers, but utilize a different mechanism than Skyplane.

There have been decades of improvements and optimizations at the transport layer to make TCP perform better in large-BDP settings within TCP itself [2, 15, 17, 33], while others concern operating system support for TCP [20, 24, 48]. Improvements to TCP are complementary to Skyplane. CodedBulk [61] uses network coding to complete bulk-transfer multicast jobs quickly [61]. Another set of research [18,63,64] investigates how to schedule urgent and non-urgent bulk transfers to meet a transfer's deadline. None of these techniques consider the cost of transferring data in the cloud.

Traffic engineering (TE) systems, like Google's B4 [35,36] and BwE [43] and Microsoft's SWAN [34], Cascara [57], and BlastShield [42], are used internally by cloud providers to navigate the cost-performance trade-off in their wide-area networks. The precise nature of the trade-off differs from Skyplane in two ways. First, TE systems consider costs in terms of the *bandwidth* provisioned (e.g., the cost of installing long-distance cables [36], or the 95th percentile bandwidth for peering links [57]). In contrast, Skyplane considers cost from the perspective of a cloud customer, where the cost depends on the volume and not bandwidth of data transferred. Second, TE systems like Cascara [57] assume a static topology and aim to reallocate bandwidth to save cost, with a global view of a single provider's network. Skyplane optimizes a single user's transfer, with the ability to use overlay regions in multiple cloud providers' networks.

Skyplane has similarities to Content Delivery Networks (CDNs) [58], most notably in that both make use of overlay networks. However, Skyplane's focus is different from CDNs. CDNs focus on caching objects near users, in order to provide low network latency. In contrast, Skyplane focuses on transferring large amounts of data quickly, with a focus on achieving high bandwidth rather than low network latency such as in workloads like ML training and database replication. CDNs are more suitable for workloads where popular objects need to be replicated to many regions so that geo-distributed users can access them with low network latency.

One application of bulk transfers is VM migration [19, 32, 37, 45] that balance VM downtime and bandwidth consumed when transferring VMs. Supercloud [37] uses a network of vSwitches in an overlay that maintains TCP connections upon migration, not to provide high bandwidth at low cost.

Some existing research efforts and commercial products focus on bulk transfer jobs that are not time-critical. For example, Laoutaris et al. [47] propose techniques to reduce the cost of transferring data for delay tolerant applications.

Cloud providers provide services for bulk transfer, such as AWS Snowball [62], Azure Data Box [12], and GCP Transfer Appliance [21], that have users ship their data via physical drives via the postal service. For sufficiently large transfers, these services may allow data to be transferred into the cloud datacenter more quickly than using the Internet.

9 Conclusion

This paper explores how to efficiently transfer data between cloud regions using cloud-aware overlay networks. Our key observation is that principles from overlay networks can be applied to the cloud setting to identify high-quality network paths that lead to fast transfer times. However, adapting principles from overlay networks to the cloud setting requires consideration of cloud resource pricing, most notably the egress fees associated with network bandwidth. Skyplane manages the trade-off between performance and cost when performing bulk data transfer. It works by accepting a user- or application-provided constraint on performance and solving a mixed integer linear program (MILP) to obtain the optimal data transfer plan. Skyplane can reduce the time to transfer data by up to 5.0× at minimal additional cost.

Acknowledgments

We thank the anonymous reviewers and our shepherd, Rachee Singh, for their helpful feedback. We also thank Asim Biswal, Jason Ding, Daniel Kang, Vincent Liu, Xuting Liu, and Anton Zabreyko. This work is supported by NSF CISE Expeditions Award CCF-1730628, NSF GRFP Award DGE-1752814, and gifts from Amazon, Astronomer, Google, IBM, Intel, Lacework, Microsoft, Nexla, Samsung SDS, and VMWare.

References

[1] William Allcock, John Bresnahn, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. The Globus striped GridFTP framework and server. In *Supercomputing*, 2005.

[2] M. Allman, D. Glover, and L. Sanchez. Enhancing TCP over satellite channels using standard mechanisms. RFC 2488, 1999.

[3] Amazon Web Services. Amazon Bottlerocket OS. https://aws.amazon.com/bottlerocket, 2022.

[4] Amazon Web Services. Amazon EC2 instance network bandwidth. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-network-bandwidth.html, 2022.

[5] Amazon Web Services. Aws DataSync: online data transfer and migration. https://aws.amazon.com/datasync, 2022.

[6] Amazon Web Services. EC2 on-demand instance pricing. https://aws.amazon.com/ec2/pricing/on-demand/, 2022.

[7] Amazon Web Services. General purpose instances. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/general-purpose-instances.html#general-purpose-network-performance, 2022.

[8] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *SOSP*. ACM, 2001.

[9] Michael Armbrust, Ali Ghodsi, Reynold Xin, and Matei Zaharia. Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics. In *CIDR*, 2021.

[10] GCT authors. Grid community toolkit. https://github.com/gridcf/gct, 2022.

[11] Microsoft Azure. Copy blob from URL. https://docs.microsoft.com/en-us/rest/api/storageservices/copy-blob-from-url, 2022.

[12] Microsoft Azure. Microsoft Azure Data Box. https://azure.microsoft.com/en-us/products/databox/, 2022.

[13] Microsoft Azure. Scalability and performance targets for blob storage. https://learn.microsoft.com/en-us/azure/storage/blobs/scalability-targets, 2022.

[14] Aran Bergman, Israel Cidon, Isaac Keslassy, Noga Rotman, Michael Schapira, Alex Markuze, and Eyal Zohar. Pied Piper: Rethinking Internet data delivery. In *CoRR*, 2018.

[15] D. Borman, B. Braden, and V. Jacobson. TCP extensions for high performance. RFC 7323, 2014.

[16] Chris X. Cai, Franck Le, Xin Sun, Geoffrey G. Xie, Hani Jamjoom, and Roy H. Campbell. CRONets: Cloud-routed overlay networks. In *ICDCS*. IEEE, 2016.

[17] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: congestion-based congestion control. In *CACM*, 2017.

[18] Bin Bin Chen and Pascale Vicat-Blane Primet. Scheduling deadline-constrained bulk data transfers to minimize network congestion. In *CCGrid*. IEEE, 2007.

[19] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *NSDI*, 2005.

[20] David D. Clark. The structuring of systems using upcalls. In *SOSP*, 1985.

[21] Google Cloud Platform. Google Cloud transfer appliance. https://cloud.google.com/transfer-appliance/docs/4.0/overview, 2022.

[22] AzCopy contributors. Azure storage AzCopy. https://github.com/Azure/azure-storage-azcopy, 2022.

[23] TensorFlow contributors. Training ResNet on Cloud TPU. https://cloud.google.com/tpu/docs/tutorials/resnet, 02 2022.

[24] Peter Druschel and Larry L. Peterson. Fbufs: A high-bandwidth cross-domain transfer facility. In *SOSP*, 1993.

[25] Sally Floyd and Kevin Fall. Promoting the use of end-to-end congestion control in the Internet. *Trans. Networking*, 1999.

[26] Forrester/Virtustream. A clear multicloud strategy delivers business value.

[27] Sebastian Frischbier, Alessandro Margara, Tobias Freudenreich, Patrick Eugster, David Eyers, and Peter Pietzuch. McCAT: Multi-cloud cost-aware transport. In *EuroSys Poster Track*, 2014.

[28] Jim Gettys and Kathleen Nichols. Bufferbloat: Dark buffers in the Internet. *CACM*, 2012.

[29] Google Cloud. All networking pricing | Virtual Private Cloud | Google Cloud. https://cloud.google.com/vpc/network-pricing, 2022.

[30] Google Cloud. Network bandwidth | Compute Engine Documentation | Google Cloud. https://cloud.google.com/compute/docs/network-bandwidth, 2022.

[31] Google Cloud Platform. Storage transfer service. https://cloud.google.com/storage-transfer-service, 2022.

[32] Kiryong Ha, Yoshihisa Abe, Thomas Eiszler, Zhuo Chen, Wenlu Hu, Brandon Amos, Rohit Upadhyaya, Padmanabhan Pillai, and Mahadev Satyanarayanan. You can teach elephants to dance: Agile VM handoff for edge computing. In *SEC*, 2017.

[33] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A new TCP-friendly high-speed TCP variant. In *SIGOPS*, 2008.

[34] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nandury, and Roger Wattenhofer. Achieving high utilization with software-driven WAN. In *SIGCOMM*, 2013.

[35] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Kondapa Naidu B., Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, Steve Padgett, Faro Rabe, Saikat Ray, Malveeka Tewari, Matt Tierney, Monika Zahn, Jonathan Zolla, Joon Ong, and Amin Vahdat. B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN. In *SIGCOMM*, 2018.

[36] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla, Us Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a globally-deployed Software Defined WAN. In *SIGCOMM*, 2013.

[37] Qin Jia, Zhiming Shen, Weijia Song, Robbert van Renesse, and Hakim Weatherspoon. Supercloud: Opportunities and challenges. In *SIGOPS*, 2015.

[38] Srikanth Kandula, Ishai Menache, Roy Schwartz, and Spandana Raj Babbula. Calendaring for wide area networks. In *SIGCOMM*. ACM, 2014.

[39] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *STOC*, 1984.

[40] Gaurav Khanna, Umit Catalyurek, Tahsin Kurc, Rajkumar Kettimuthu, P. Sadayappan, Ian Foster, and Joel Saltz. Using overlays for efficient data transfer over shared wide-area networks. In *Supercomputing*, 2008.

[41] Dejan Kostić, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. *SOSP*, 2003.

[42] Umesh Krishnaswamy, Rachee Singh, Nikolaj Bjørner, and Himanshu Raj. Decentralized cloud wide-area network traffic engineering with BlastShield. In *NSDI*. USENIX, 2022.

[43] Alok Kumar, Sushant Jain, Uday Naik, Anand Raghuraman, Nikhil Kasinadhuni, Enrique Cauich Zermeno, C. Stephen Gunn, Jing Ai, Björn Carlin, Mihai Amarandei-Stavila, Mathieu Robin, Aspi Siganporia, Stephen Stuart, and Amin Vahdat. BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing. In *SIGCOMM*, 2015.

[44] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*, chapter 3, page 308. International 6th edition, 2013.

[45] H. Andrés Lagar-Cavilla, Joseph A. Whitney, Roy Bryant, Philip Patchin, Michael Brudno, Eyal de Lara, Stephen M. Rumble, M. Satyanarayanan, and Adin Scannell. Snowflock: Virtual machine cloning as a first-class cloud primitive. *ACM Trans. Comput. Syst.*, 29(1), feb 2011.

[46] Fan Lai, Mosharaf Chowdhury, and Harsha Madhyastha. To relay or not to relay for Inter-Cloud transfers? In *HotCloud*, 2018.

[47] Nikolaos Laoutaris, Georgios Smaragdakis, Pablo Rodriguez, and Ravi Sundaram. Delay tolerant bulk data transfers on the Internet. In *SIGMETRICS*, 2009.

[48] Chris Maeda and Brian N. Bershad. Protocol service decomposition for high-performance networking. In *SOSP*, 1993.

[49] Miguel Matos, António Sousa, José Pereira, and Rui Oliveira. CLON: Overlay network for clouds. In *WDDM*, 2009.

[50] Microsoft Azure. Scalability and performance targets for blob storage. https://docs.microsoft.com/en-us/azure/storage/blobs/scalability-targets, 2021.

[51] Microsoft Azure. Pricing - bandwidth | Microsoft Azure. https://azure.microsoft.com/en-us/pricing/details/bandwidth/, 2022.

[52] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The Akamai network: A platform for high-performance Internet applications. *SIGOPS*, 2010.

[53] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling tcp throughput: A simple model and its empirical validation. In *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '98, page 303–314, New York, NY, USA, 1998. Association for Computing Machinery.

[54] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. Low latency geo-distributed data analytics. In *SIGCOMM*, 2015.

[55] Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek S. Pai, and Michael J. Freedman. Aggregation and degradation in JetStream: Streaming analytics in the wide area. In *NSDI*, 2014.

[56] Noga H. Rotman, Yaniv Ben-Itzhak, Aran Bergman, Israel Cidon, Igor Golikov, Alex Markuze, and Eyal Zohar. CloudCast: Characterizing public clouds connectivity. *CoRR*, 2022.

[57] Rachee Singh, Sharad Agarwal, Matt Calder, and Paramvir Bahl. Cost-effective cloud edge traffic engineering with CASCARA. In *NSDI*, 2021.

[58] Ramesh K. Sitaraman, Mangesh Kasbekar, Woody Lichtenstein, and Manish Jain. Overlay networks: An Akamai perspective. *Advanced Content Delivery, Streaming, and Cloud Services*, 2014.

[59] H. Sivakumar, S. Bailey, and R. L. Grossman. PSockets: The case for application-level network striping for data intensive applications using high speed wide area networks. In *Supercomputing*. ACM/IEEE, 2000.

[60] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *SIGCOMM*, 2001.

[61] Shih-Hao Tseng, Saksham Agarwal, Rachit Agarwal, Hitesh Ballani, and Ao Tang. CodedBulk: Inter-datacenter bulk transfers using networkcoding. In *NSDI*, 2021.

[62] Amazon Web Services. AWS Snowball. https://aws.amazon.com/snowball, 2022.

[63] Yu Wu, Zhizhong Zhang, Chuan Wu, Chuanxiong Guo, Zongpeng Li, and Francis C. M. Lau. Orchestrating bulk data transfers across geo-distributed datacenters. *Trans. Cloud Computing*.

[64] Hong Zhang, Kai Chen, Wei Bai, Dongsu Han, Chen Tian, Hao Wang, Haibing Guan, and Ming Zhang. Guaranteeing deadlines for inter-datacenter transfers. In *EuroSys*, 2015.

[65] Yuchao Zhang, Junchen Jiang, Ke Xu, Xiaohui Nie, Martin J. Reed, Haiyang Wang, Guang Yao, Miao Zhang, and Kai Chen. BDS: A centralized near-optimal overlay network for inter-datacenter data replication. In *EuroSys*, 2018.