Cloudcast: High-Throughput, Cost-Aware Overlay Multicast in the Cloud

Sarah Wooders and Shu Liu, *UC Berkeley;* Paras Jain, *Genmo AI;* Xiangxi Mo and Joseph E. Gonzalez, *UC Berkeley;* Vincent Liu, *University of Pennsylvania;* Ion Stoica, *UC Berkeley*

https://www.usenix.org/conference/nsdi24/presentation/wooders

This paper is included in the
Proceedings of the 21st USENIX Symposium on
Networked Systems Design and Implementation.

April 16–18, 2024 • Santa Clara, CA, USA

978-1-939133-39-7

Cloudcast: High-Throughput, Cost-Aware Overlay Multicast in the Cloud

Sarah Wooders
UC Berkeley

Shu Liu
UC Berkeley

Paras Jain
Genmo AI

Xiangxi Mo
UC Berkeley

Joseph E. Gonzalez
UC Berkeley

Vincent Liu
University of Pennsylvania

Ion Stoica
UC Berkeley

Abstract

Bulk data replication across multiple cloud regions and providers is essential for large organizations to support data analytics, disaster recovery, and geo-distributed model serving. However, data multicast in the cloud can be expensive due to network egress costs and slow due to cloud network constraints. In this paper, we study the design of high-throughput, cost-optimized overlay multicast for bulk cloud data replication that exploits trends in modern provider pricing models along with techniques like ephemeral waypoints to minimize cloud networking costs.

To that end, we design an optimization algorithm that uses information about cloud network throughput and pricing to identify cost-minimizing multicast replication trees under user-given runtime budgets. Our open-source implementation, Cloudcast, is used for cloud overlay multicast that supports pluggable algorithms for determining the multicast tree structure. Our evaluations show that Cloudcast achieves 61.5% cost reduction and 2.3× replication speedup compared to both academic and commercial baselines (e.g., AWS multi-region bucket) for multi-region replication.

1 Introduction

Increasingly, data in the cloud must be replicated to multiple cloud providers and different regions within each provider. For example, geo-distributed applications like model serving require model weights or features computed in a single region to be replicated to multiple geographic regions to reduce serving latency for users accros the globe [22,44,51]. Data sharing between collaborating organizations using different providers similarly requires replicating data to multiple locations. Finally, the growth of multi-cloud applications that leverage resources from multiple providers is dependent on application data being available across provider boundaries [13,51,52].

Of course, data replication and multicast are not new. Both topics have been extensively studied to optimize throughput and scalability in the context of IP networks, peer-to-peer overlays [12,14,19,22,33], and inter-DC networks [20,36,47,54].

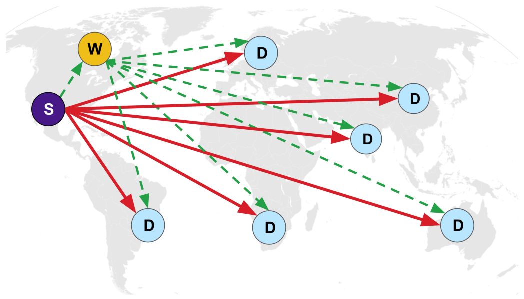


Figure 1: Direct replication from a source region (purple) to destination regions (blue) may traverse expensive or slow links, which can be avoided via *waypoint* regions (yellow).

However, replication between cloud regions and providers introduces first-order concerns beyond just throughput and scalability. In particular, the *monetary cost* of the transfer is a critical factor and one that (as we show later in this paper) is poorly handled by existing techniques for optimizing throughput [33,36,54]. While some existing works consider the monetary cost for multicast, they either ignore the throughout [24] or assume a capacity-based pricing model [35] which is inconsistent with today's cloud. In contrast to capacity-based pricing, cloud providers charge per-GB network egress fees for data transferred out of a given region to another region or cloud provider. Per-GB egress fees introduce a multiplicative term into the transfer cost—(egress price)×(amount transferred)—making it significantly more difficult to optimize throughput and cost.

Egress costs can vary by orders of magnitude depending on the source and destination [42], as well as the capacity of cross-region links. As a result, the structure of the multicast replication tree (i.e., what data is replicated along which paths) can dramatically affect the end-to-end throughput and monetary cost of replication. As a concrete example, consider replication from a GCP source region to six AWS regions (Figure 1). Direct replication of the data between the source and each destination region (shown in red arrows) would cost $720 per TB. Instead, replicating to an AWS region with the lowest cross-region egress fees once and multicasting data from that AWS region to other regions (shown in dotted green arrows) would reduce the price to $240 per TB. Further modifying the multicast tree to utilize high-throughput links and offload egress bandwidth from the source node can also improve throughput.

In this work, we solve the problem of *high-throughput, cost-optimized cloud multicast* in which we minimize the cost of data replication while achieving a target replication time (across all destinations) for bulk multicast replication. Cloud multicast incurs costs from network egress fees and compute resources needed to mediate the transfer. In addition, cloud multicast must meet application Service Level Objectives (SLO) for the replication time, such as providing freshness guarantees on replicated data.

We design an optimizer to determine a multicast tree structure given a user-specified source region, destination regions, and target replication time. By providing varying target replication times, our optimizer can generate a Pareto-curve (shown in Figure 8) that improves replication cost and throughput compared to prior approaches for cloud multicast [23, 24]. We achieve this by leveraging techniques such as striping, VM parallelism, and overlay networking, while also accounting for the cloud providers' network characteristics, resource constraints, and per-GB network pricing model.

Designing this optimization is challenging for two main reasons. First, the optimizer must account for path-specific pricing models, resource constraints, and varying performance across cloud providers. Existing techniques that formulate the optimization problem in terms of bandwidth allocation cannot be adapted to account for per-GB network pricing without making the problem non-linear (described further in Section 3). Second, the optimization search space is combinatorially large, as the optimizer must determine both the set of overlay waypoint regions (regions which are neither the source nor destination) as well as how data will be routed along the overlay network. Unlike the traditional overlay settings, the cloud offers significantly more flexibility in the number and the location of overlay nodes, as cloud VMs can dynamically be instantiated in specified cloud provider regions. Furthermore, replicating subsets of data (i.e., stripes) via different paths is critical for achieving high-throughput [12]. We introduce several approximations (e.g., pre-selecting the regions and limiting path lengths) to reduce this search space and enable the optimizer to run within seconds.

To run overlay multicast across clouds, we develop *Cloudcast*, a system for bulk data overlay multicast across GCP, AWS, and Azure. Cloudcast has a centralized control plane that supports pluggable algorithms for determining the number and location of overlay nodes and replication trees for multiple segments of data. We implement our optimizer as well as several baseline algorithms as part of Cloudcast's control plane. We run system experiments to multicast data across clouds and show that Cloudcast is able to achieve up to 62.4% cost savings and 2.84× replication speedup depending on the

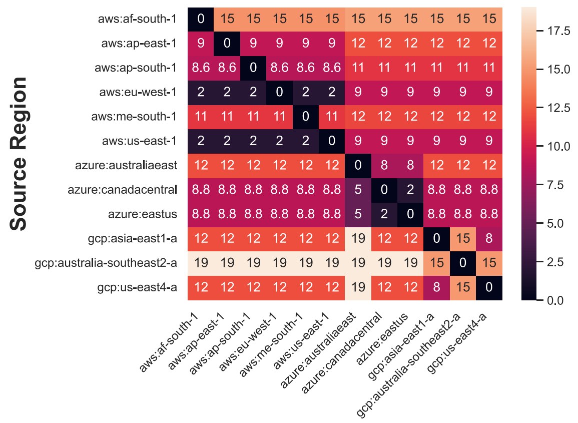


Figure 2: Egress fees between regions (in cents per GB).

control plane algorithm (Figure 10).

We run an end-to-end system evaluation comparing Cloudcast with BitTorrent [19] and AWS's commercial offering for multi-region bucket replication [50], which, like most cloud data replication offerings, only supports replication into or within that cloud. We find that Cloudcast achieves 7.7× replication speedups and 28.4% cost savings compared to BitTorrent (Figure 12). Compared to multi-region bucket replication, we find that Cloudcast achieves up to 61.5% cost reduction and 2.3× replication speedup (Figure 11).

To summarize, we make the following contributions:

1. We design an optimizer for minimizing replication cost under replication time constraints.
2. We introduce several approximations to reduce the search space for the optimizer, reducing the solver runtime from hours to seconds.
3. We build Cloudcast, an open-source system for cloud overlay multicast with pluggable data transfer policy.

2 Problem Setup

We frame the problem of cloud multicast in terms of constructing an overlay network for replicating data, which involves defining: (1) the set of *overlay nodes* (i.e., cloud VMs) and (2) the *paths* between those nodes that will be included in a multicast replication tree. Cloudcast eventually divides the target data into multiple *stripes* (i.e., partitions), so concurrent replication trees may be used in a single transfer. Our optimization objective is to minimize the monetary cost of replication while also meeting a replication time constraint.

2.1 Egress Costs

A unique aspect of multicast in the cloud is the effect of egress costs incurred for data transferred across cloud regions. Cloud providers charge for wide-area data transfer *per-GB* of data transferred. Egress prices—as a method of keeping data within the provider's regions without disincentivizing

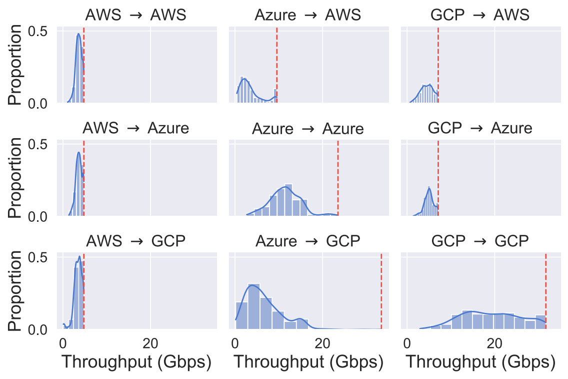


Figure 3: Bandwidth distribution (in Gbps) between regions. Per-VM egress limits are marked in red dotted lines.

migration into the provider—dominate data movement costs in the cloud and fundamentally change the multicast problem. Figure 2 visualizes the pricing for 11 regions across AWS, Azure, and GCP. Prices vary depending on the source and destination cloud or region, with differences of up to 23× across region pairs. Along those lines, one particularly important axis is whether the transfer stays within a given cloud provider or crosses provider boundaries, as inter-cloud egress costs are generally higher than intra-cloud egress.

Intra-cloud egress (data movement between geographically separated datacenters in the same cloud provider) is priced between \$0.01 – \$0.19 per GB transferred. Prices typically increase with longer-distance transfers. For example, GCP charges \$0.08 for transfers between continents but only \$0.02 for transfers within the US. Some smaller providers (e.g., IBM, Cloudflare) offer free cross-region egress.

Inter-cloud egress (data movement between different cloud providers) is typically priced at a much higher rate per GB (\$0.08 – \$0.23). As such, it is essential to minimize cross-cloud transfers in a multicast replication tree.

2.2 Bandwidth Variability Across Endpoints

Meeting replication time constraints can be challenging due to network bandwidth variability in the cloud. One type of variability arises from cloud providers, who impose constraints on per-VM egress and ingress bandwidth. These constraints differ significantly across providers: for instance, AWS throttles intra-cloud and inter-cloud egress to 5 Gbps per VM, while Azure imposes no VM-level limits. The impact of these egress limits can be observed in Figure 3, where bandwidth is capped at the VM egress limit for AWS and GCP. Limited node egress poses a particular challenge for cloud multicast, as the source node's egress bandwidth is often the bottleneck.

Even when source-node bandwidth is not the bottleneck, observed network capacity can also vary considerably across cloud region pairs (up to 202×). Note that these networks are relatively stable across time; prior work [28] has found that network throughputs are stable over periods of at least 24 hours. Instead, variations are primarily observed across different source and destination regions. Figure 3 depicts the distribution of profiled bandwidth between VMs running in AWS, Azure, and GCP. Intra-cloud bandwidth is typically (but not always) higher than inter-cloud bandwidth.

2.3 Elasticity of Resources

A major advantage of the cloud is resource elasticity and the ability to flexibly provision VMs across many regions. In the face of the source bottlenecks described above, VM elasticity translates to a corresponding *elasticity of bandwidth*. Allocating multiple parallel VMs enables users to scale throughput beyond per-VM network bandwidth limits.

Unfortunately, adding elastic VM capacity at the source region has limitations. Additional VMs add additional costs due to per-second billing on VMs, which can impact the cost/throughput tradeoff. We note that because the marginal cost of additional VMs is often relatively small compared to egress fees, the tradeoff is often worth making. However even in these situations, bandwidth elasticity has limits: for instance, if a network-based bottleneck is unavoidable or when cloud providers limit the number of vCPUs per region.

Crucially, elastic VM capacity can also be deployed at *waypoint regions* that are neither the source nor the destination. These waypoint regions can help mitigate source VM bottlenecks by distributing load from multicast fan-out across multiple separate regions. Waypoint regions also mitigate points of congestion by routing data around slow paths.

2.4 Illustrated Example

Selecting overlay nodes and replication trees to optimize cost and throughput is challenging. Consider the toy example in Figure 4 for a 2 GB replication with two 1 GB stripes. Assuming a 4 Gbps bandwidth limit for all nodes and one VM per region, the source ("S") and destination ("D") nodes have fast but expensive outgoing paths, capable of sending at 2 Gbps but costing 10¢ per GB transferred. Other regions have cheaper but slower outgoing paths, capable of sending at 1 Gbps but costing 2¢ per GB transferred. In a simple direct replication scenario, the replication will be bottlenecked by the source node's egress limit (4 Gbps). With two copies of data to send, the total transfer time will be 8 seconds.

Like many bandwidth-optimized techniques [12, 23, 33], we offload egress bandwidth by sending a single data copy from the source and leveraging multiple replication trees. Replication cost is reduced by replicating to a waypoint, and then multicasting to destinations. This doubles replication time to 16 seconds due to stripes being replicated via the slower path (dotted arrows). An 8-second replication SLO is met by transferring just one stripe via the cheaper waypoint.

This simple example presents a large search space for possible replication trees, and real-world cloud networks present

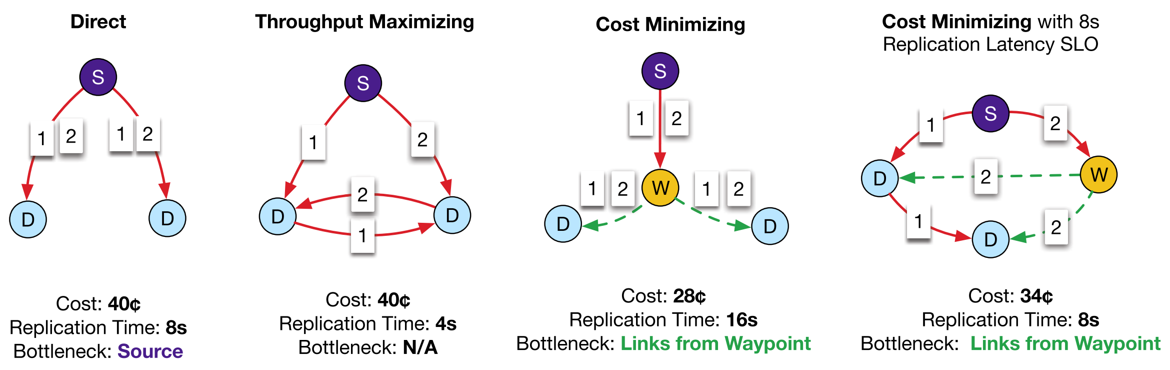


Figure 4: Overlay node set and distribution trees for a toy example. The source and destination nodes are marked 'S' and 'D' respectively, while waypoint nodes in yellow are marked 'W'. Expensive, fast paths (\$0.1 per GB, 2 Gbps) are shown in solid red, while slow, cheap paths (\$0.02 per GB, 1 Gbps) are shown in dashed green.

additional parameters such as choosing the number of VMs per region and many possible waypoint regions.

3 Cost Optimization in Cloudcast

We design an optimizer to minimize replication cost while meeting a replication time SLO (i.e., a constraint on the maximum replication time to a destination). Our optimizer has two main contributions. The first is a Mixed-Integer Linear Program (MILP) formulation of the cost-aware multicast problem, jointly selecting overlay nodes and replication trees. While others [23, 54] have used MILP formulations for multicast overlay design, they formulate the optimization problem in terms of *bandwidth*. Extending these formulations to accommodate per-GB costs would violate linearity as data transfer volume (cost) is proportional to the product of the key decision variables: allocated bandwidth and replication time. As a consequence, we propose a new formulation that reframes the optimization in terms of data volume. Our new formulation assigns discrete subsets of data (i.e. stripes) to replication paths in the network while ensuring that a complete copy of the data arrives at all destinations. Unfortunately, solving this MILP formulation can be intractable for larger numbers of destinations. Our second contribution is an approximation of the MILP formulation that significantly reduces solve time without significantly degrading the solution quality.

3.1 Egress Cost Minimization Algorithms

The challenge with our optimization problem stems from having to consider *both* throughput and cost. Without replication time constraints, we observe that the Steiner Tree [27] minimizes egress cost. A Steiner Tree is a set of cost-minimizing edges that form a tree that connects a subset of nodes within a graph. If we do not allow the use of waypoint regions, the cost-minimizing tree is a Minimum Spanning Tree (MST). While solving for the MST can be done in linear time, the Steiner Tree problem is NP-hard, though many approximations exist [43]. We cannot use the Steiner Tree to account

Inputs	
$\text{TRANSFER-SIZE} \in \mathbb{R}$	*Transfer size in GB*
$\text{TIME} \in \mathbb{R}$	*Replication time constraint*
$\text{STRIPES} \in \mathbb{Z}_+$	*Number of data stripes*
Decision Variables	
$P \in \{0,1\}^{\lvert\text{STRIPES}\rvert \times \lvert V\rvert \times \lvert V\rvert}$	*Path indicator variable*
$N \in \mathbb{Z}_+^{\lvert V\rvert}$	*Number of VMs per region*
$F \in \mathbb{R}_+^{\lvert\text{STRIPES}+1\rvert \times \lvert V\rvert \times \lvert V\rvert}$	*Flow feasibility variable*
Constants: Cross-Region Paths (edges)	
$\text{BANDWIDTH}^{path} \in \mathbb{R}_+^{\lvert V\rvert \times \lvert V\rvert}$	*Bandwidth profile matrix (Gbps)*
$\text{COST}^{path} \in \mathbb{R}_+^{\lvert V\rvert}$	*Network cost (\$/Gbit)*
Constants: VM Instances (nodes)	
$\text{EGRESS}^{VM} \in \mathbb{R}_+^{\lvert V\rvert}$	*Per region per VM egress limit (Gbps)*
$\text{INGRESS}^{VM} \in \mathbb{R}_+^{\lvert V\rvert}$	*Per region per VM ingress limit (Gbps)*
$\text{COST}^{VM} \in \mathbb{R}_+^{\lvert V\rvert}$	*Per region per VM cost (\$/s)*
$\text{LIMIT}^{VM} \in \mathbb{Z}_+^{\lvert V\rvert}$	*Max number of VMs per region*

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

for replication throughput or instance costs, since it only optimizes total edge cost, but we expect our optimizer's solution to be similar to a Steiner Tree in cases where the replication time constraint is loose.

3.2 Profiling Cross-region Bandwidth

The bandwidth of paths between cloud regions (both intra-cloud and inter-cloud) is determined by the number of VMs in each region, each VM's egress and ingress limits, and the profiled bandwidth. As discussed in Section 2.2, cross-region bandwidth per VM can be estimated by profiling the bandwidth between region pairs using `iperf3`. Egress and ingress limits vary across cloud providers but are static and can be determined by cloud providers' documentation [8, 10, 16]. We utilize these profiles as an estimate of expected network bandwidth for the duration of a transfer. Profiling results are included as part of our open-source repository and shared across all users of Cloudcast.

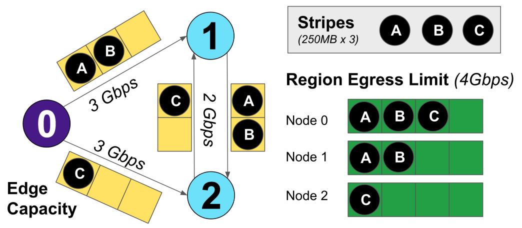


Figure 5: Stripes transferred from the source (purple) to destinations (blue) are placed by the solver along edges depending on edge capacity (yellow) and node capacity (green).

3.3 Optimizing Cost with Time Constraints

In order to minimize replication price while meeting runtime requirements and cloud resource constraints, we frame a MILP on a directed graph representing the entire cloud topology. The input to the optimizer is the transfer size: TRANSFER-SIZE, the runtime budget: TIME, and the number of stripes: STRIPES to divide the data into.

To formulate the optimization problem as a MILP, formulate the problem in terms of allocating data *volume* to edges rather than bandwidth, with allocation units per stripe. We translate cross-region bandwidth and per-region egress/ingress limits into volume capacities, as shown in Figure 5, this determines how many stripes can fit along each edge. This makes the MILP similar to a bin packing problem, where we aim to pack stripes into edges such that all destinations receive all stripes. The volume-based representation allows cost to be computed as a function of the number of stripes placed on each edge.

Next, we formally describe the MILP decision variables, objectives, and constraints. The cloud regions and cross-region paths are represented as $G = (V, E)$, where V denotes the set of cloud regions and E denotes paths between regions. We provide a reference table for the notation in Table 1.

3.3.1 Decision variables

The MILP formulation consists of three decision variables. The path indicator variable $P_{s,(v,u)}$ indicates whether a stripe s is sent between regions $(u, v) \in V$. The paths selected by P make up the multicast replication tree for each stripe. The decision variable N_v represents the total number of overlay routers in the region v. An additional flow variable $F_{s,(u,v)}$ ensures valid paths when constructing the multicast tree. It ensures that the paths selected by P do not contain cycles and are connected, by allowing flow to be pushed from the source to all destinations for each stripe (see Section 3.3.3).

3.3.2 Objective: minimizing price under a deadline

To minimize the price of a multicast transfer while meeting replication time constraints, we use a two-part objective function. The first part optimizes the number of virtual machines (VMs) per region, represented by N, and the second part optimizes the distribution trees per stripe, represented by P. The objective is formulated as follows:

$$\arg\min_{P,N} \underbrace{\text{TIME} \times \langle \text{COST}^{VM}, N \rangle}_{\text{Instance Cost}} \tag{1}$$

$$+ \underbrace{\frac{\text{TRANSFER-SIZE}}{\text{STRIPES}} \times \sum_{s \in \text{STRIPES}} \langle \text{COST}^{path}, P_s \rangle}_{\text{Egress Cost}} \tag{2}$$

The price of a data transfer is the sum of the instance fee and the egress fee. The instance fee depends on the number of VMs running per region, the job completion time, and the per-region VM fee. The egress fees are determined by the data distribution path and the amount of data traversed through the path, as defined by P. We note that the instance cost is also an upper bound as it can be potentially overestimated if the data transfer is completed in less than the user-defined time budget. However, this is necessary to ensure linearity.

3.3.3 Constraints

We represent cross-region bandwidth, node egress/ingress bandwidth, per-region VM limits, and replication tree structure requirements as constraints within the MILP.

Representing Inter-Region & Inter-Cloud Bandwidth. Cross-region bandwidth is represented as the *per-GB capacity* given the run-time budget, i.e., how many stripes can fit along an edge. Increasing the number of VMs in the source regions linearly increases the rate at which we can send data. We thus model the bandwidth between two regions as the per-VM bandwidth profiled between those two regions multiplied by the number of VMs in the source region:

$$\text{CAPACITY}^{path} = \langle N, \text{BANDWIDTH}^{path} \rangle * \text{TIME}, \tag{3}$$

and constrain P in terms of the path capacity:

$$\forall (u,v) \in E \qquad \text{SIZE}_{\text{STRIPE}} * \sum_s P_{s,(u,v)} \leq \text{CAPACITY}^{path}{}_{(u,v)}. \tag{4}$$

to ensure allocated stripes fit within the capacity.

Representing VM Bandwidth Constraints. Cloud providers impose per-VM bandwidth constraints on network egress, as described in Section 2.2. As such, a major bottleneck of multicast transfer is the source region's limited egress bandwidth. We constrain P in terms of the ingress and egress limits:

$$\forall v \in V \;\; \text{SIZE}_{\text{STRIPE}} * \sum_s \sum_{u \in V} P_{s,(v,u)} \tag{5}$$

$$\leq \text{EGRESS}^{VM}{}_v * N_v * \text{TIME} \tag{6}$$

$$\forall u \in V \;\; \text{SIZE}_{\text{STRIPE}} * \sum_s \sum_{v \in V} P_{s,(v,u)} \tag{7}$$

$$\leq \text{INGRESS}^{VM}{}_u * N_u * \text{TIME} \tag{8}$$

Representing VM Capacity Constraints. We account for per region VM limits by adding the constraint $N \leq \text{LIMIT}^{VM}$.

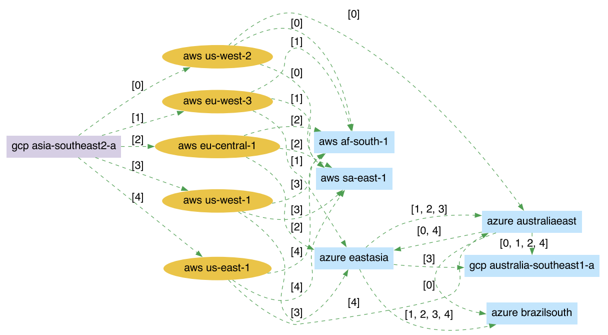


Figure 6: Visualized solver output for inter-cloud replication described in Section 5.1, consisting of source (purple), waypoint (yellow), and destination (blue) regions. The data is divided into 5 stripes (marked on edges).

Ensuring Valid Multicast Trees. We use an additional variable F to ensure that the paths selected by P are valid distribution trees, i.e., they are connected and acyclic, and they deliver all data to each destination. At a high level, we ensure that $F_{s,(u,v)} \geq 1$, if $P_{s,(u,v)} = 1$, and impose conservation of flow constraints on F but not P, since P is an indicator variable not a flow variable. We then ensure that flow can be pushed from the source node to destination nodes on F for each stripe, which also ensures that flow can be pushed from the source to destination for the paths selected by P (without having to impose flow conservation on P). We leave details on this part of the formulation for Appendix B due to space.

3.3.4 Solver feasibility

Our formulation so far has a search space of size $O(2^{|V|^2 \times |\text{STRIPES}|})$. With 71 possible regions across GCP, AWS, and Azure and 10 stripes, the search space is, therefore, $O(2^{50410})$, which is infeasible even for advanced solvers to solve within a few minutes, necessitating approximations.

3.4 Reducing Optimizer Runtime

In this section, we describe several mechanisms that we combine to reduce the optimization runtime or an order of seconds, while still maintaining solution quality.

Node Clustering. We observe that many regions across cloud providers share similar characteristics in terms of bandwidth and the costs of their outgoing and incoming paths. A motivating observation was that sub-sampling regions randomly could produce similar solutions with much lower solve time, as shown in Figure 17. At a high level, AWS regions in Europe regions all have similar egress/ingress costs and bandwidth, so only one of those regions needs to be considered as a potential waypoint. Therefore, to reduce the optimizer search space, we cluster regions using their incoming and outgoing path costs and bandwidth as features and select a representative node from each cluster. We empirically find that, with about 20 clusters (i.e. 20 subsampled regions), the optimizer can generate solutions that are reliably similar to the original MILP without approximation (more discussion in Section 5.3.2).

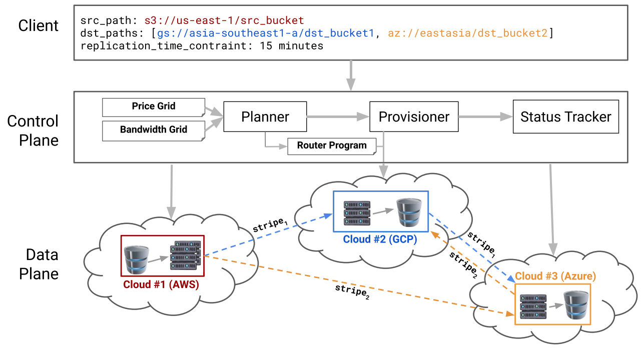

```
src_path: s3://us-east-1/src_bucket
dst_paths: [gs://asia-southeast1-a/dst_bucket1, az://eastasia/dst_bucket2]
replication_time_contraint: 15 minutes
```

Figure 7: Cloudcast system architecture.

Hop Constraining. To further reduce the optimization space, we only consider a maximum of 2-hop overlay waypoints. Previous research has shown that limited numbers of overlay hops are often sufficient [7, 41, 46]. Our analysis also found solutions using multiple overlay hops to be rare, suggesting that they need not be considered. We implement the hop constraints as an additional constraint on the MILP.

Stripe-iterative Approximation. To make the optimizer runtime linear with respect to the number of stripes (rather than exponential), we design a greedy, stripe-iterative approximation algorithm that solves for one stripe per iteration. We solve for each stripe independently, then update the input graph for the next stripe by reducing the path capacity (CAPACITY^{path}), instance limits, and egress/ingress limits per region (LIMIT^{VM}, EGRESS^{VM}, and INGRESS^{VM}).

3.5 Example Topology

We show an example of the optimizer's output replication tree topology visualized in Figure 6. Due to variability in cloud provider egress pricing and cross-region throughput, our optimizer often finds unexpected solutions, such as routing one stripe (marked `[3]`) from GCP to AWS, AWS to Azure, then back to GCP. Although questionable at first glance, we evaluate this same replication in Figure 10 and demonstrate both cost and replication time improvements over baselines.

4 Architecture of Cloudcast

A key contribution of this work is the design and implementation of the Cloudcast artifact, which provides a practical, performant, and extensible system for studying overlay multicast algorithms in cloud environments. The Cloudcast system simplifies the design and deployment of multicast overlays spanning cloud object stores, and is used to implement the optimizer described in Section 3.

We provide an overview of Cloudcast in Figure 7. Cloudcast is designed with a centralized control plane and a distributed data plane. The control plane determines the set of overlay nodes and routing paths, and it dispatches and monitors multicast jobs. The data plane consists of *overlay routers*, which we implement as modular software routers running on overlay nodes deployed on cloud VMs. The control plane configures overlay routers using a *router program*, which specifies a graph of modular operators for processing data. Each overlay router gets a unique router program, which, in cooperation with other routers in the system, implements the desired flow of data over the overlay network.

Cloudcast is implemented as part of the Skyplane [28] open source project and consisted of 5K additional lines of Python to implement a design based on overlay routers.

4.1 Control Plane

The control plane contains the planner, which supports pluggable algorithms for determining the placement of overlay routers across cloud providers and paths along which data is replicated (shown in Figure 7). The output of the planner is used to provision VMs to act as overlay routers across cloud regions and to compile a router program for each overlay router that configures its behavior. Finally, the control plane initiates the transfer and monitors its progress.

Planner. The planner is responsible for creating a multicast plan based on a target replication time, source and destination object store paths provided by the user, and profiling data described in Section 3.2. The planner takes as input the algorithm to use for generating a multicast plan, which can be the default Cloudcast optimizer described in Section 3.3 or a custom plan (e.g., a Steiner Tree over the cost graph). The planning algorithm determines how many overlay nodes to create in each region and how each data stripe should be routed through the overlay network. The planner uses the algorithm output to generate a *router program* for each overlay router, which specifies how the overlay router should process a chunk header when received. The Cloudcast default optimizer is implemented using Python's CVXPY library [3] (version 1.3.2) with a Gurobi solver [26].

Provisioner. Once a multicast plan is determined, the provisioner instantiates the overlay routers. The provisioner creates a VPC in each cloud provider and provisions VMs to act as overlay routers within these VPCs. The provisioner also sets firewall rules to allow network traffic between overlay routers, which send and receive data from each other, as specified by the planner-generated router programs. Once a VM has been instantiated, the provisioner installs and launches the router programs as containers on the VMs.

Chunk Dispatching and Status Tracker. The control plane subdivides replication target data into *chunks*, which are at most 64MB in size, to allow for transfer pipelining and parallelism. Each chunk has a *chunk header*, which specifies a key (e.g., object store object, filename), byte range, and an optional multipart ID (required for multipart uploads). The chunk header also contains a *stripe ID*, which specifies which path along the overlay the chunk will take.

The control plane informs each source overlay router (i.e., overlay routers responsible for reading source data) the chunks for which they are responsible by sending the corresponding chunk headers. We refer to this as *registering* a chunk to an overlay router. The control plane's status tracker monitors the status of each chunk by querying the status of chunks on each overlay router.

4.2 Data Plane

The data plane is composed of overlay routers, each running on a single VM. The overlay routers are created and configured by the control plane to execute the transfer according to the multicast plan. Cloudcast supports configurable overlays by defining processing on overlay routers using modular operators, inspired by the design of configurable routers [32].

The router program provided by the control plane specifies a directed acyclic graph (DAG) of *operators* (analogous to elements) and *connections*, all of which run on each overlay router and are used to process incoming chunk headers registered to the overlay router. The DAGs are created at the overlay router's startup time based on the router program, and they allow overlay routers to process chunks without additional coordination with the control plane.

Operators are implemented as a pool of worker processes running processing steps for a chunk, such as reading the chunk from the source object store, relaying the chunk to another overlay router, writing the chunk to a destination object store, or transforming the chunk data (e.g., compression or encryption). Connections pass chunk headers between operators via thread-safe queues, and can be configured to send a chunk header to one or all of multiple downstream operators.

For example, on a source overlay router, chunk registrations from the control plane will provide chunk headers to the first operator in the DAG, which downloads chunk data from an object store. All chunk data is stored in a shared memory filesystem to allow for fast access across operators. Once chunk data is downloaded, the chunk header is passed to the next operator via a connection, which runs LZ4 compression [4]) and secret key encryption [5, 18] on the chunk data. The leaf operators are 'sender' operators, which relay the chunk header and data to other overlay routers.

Chunk data is relayed between overlay routers by a 'sender' operator on the sending router and a 'receiver' operator on the receiving overlay router. When the sender operator is created, it creates parallel TCP connections which are kept open for the duration of the transfer. Before sending chunk data, the sender will attempt to register the corresponding chunk headers with the receiving overlay router to ensure it has space in its shared memory file system to write the chunk data. Once chunks are registered, the sender will send the chunk data over the TCP sockets, and the receiver will wait for the written chunk data size to match the size specified by the chunk header, before

System	Description
Direct	Data is transferred directly from the source to the destination regions.
MDST	Data is transferred along edges selected by a Minimum Directed Spanning Tree (including source and destination regions) computed from network costs.
Steiner Tree	Data is transferred along edges selected by a Steiner tree (including optional waypoint regions) computed from network costs.
SPIDER [23]	Data is transferred according to the plan generated by SPIDER, a system designed for fast bulk replication to multiple destinations.
Skyplane	Skyplane's optimizer is used to select paths for each source-destination pair, which are combined to build the distribution tree.
CloudMPCast [24]	Data is transferred over a set of cost-minimizing edges that meet a minimum bandwidth threshold.
Deadline-aware Inter-DC Multicast [30]	Data is transferred to meet deadlines in the inter-DC context according to [30]. Note that due to scalability issues, we needed to modify the candidate tree generation step to only consider a subset of waypoint regions to achieve tractable runtimes.
AWS S3 Multi-Region Bucket [9]	Vendor product that supports intra-cloud between AWS regions only. We enable Replication Time Control [38].
Bullet [33]	Data is transferred according to the plan generated by Bullet, a high-bandwidth dissemination technique using an overlay mesh.
BitTorrent [49]	Peer-to-peer protocol where peers download data from each other in a decentralized manner.
Cloudcast-Opt (HT)	Data is transferred along the highest throughput (HT) multicast tree generated by our optimizer (tightest time constraint).
Cloudcast-Opt (LC)	Data is transferred along a low cost (LC) multicast tree generated by our optimizer (relatively loose time constraint).

Table 2: All of the systems and variants we evaluate, covering a mix of academic baselines and commercial solutions.

sending chunk headers to the next operator. Successfully sent chunk data is deleted from the shared memory filesystem.

Backpressure. Connections are configured with a maximum size for the underlying queues. If the queue reaches its maximum size, the upstream operator will wait until the queue size decreases sending chunk headers to the connection.

Striping. Registered chunk headers with different stripe IDs are placed in different queues and processed by separate DAGs, so that different stripes can be routed differently.

5 Evaluation

In this section, we evaluate Cloudcast across three metrics: replication cost, replication time, and the optimizer solve time (or simply, runtime). In particular, we show that for intra-cloud and inter-cloud bulk data transfer, Cloudcast is able to achieve up to 61.5% cost improvements under a tight runtime budget when compared to academic, commercial, and open-source baselines. We also show that our approximations to reduce the optimizer solve time (as discussed in Section 3.4) are highly effective by reducing the runtime by, on average, 30.68× for 5-destination replications. To simplify evaluation, we disable compression and encryption in experiments.

The full list of evaluated baselines is shown in Table 2. We note that many algorithms do not determine the number of VMs to use in each region. To present them in the best light possible, we maximize the number of VMs in each region traversed by data, subject to per-region quota limits.

5.1 Comparison to Multicast Algorithms

We compare the replication time and cost of existing multicast algorithms with Cloudcast's optimizer to send 100 GB of data from one source to six destination regions.

Simulation results. Given the above replication scenario, we start by exploring a wide range of algorithmic baselines and

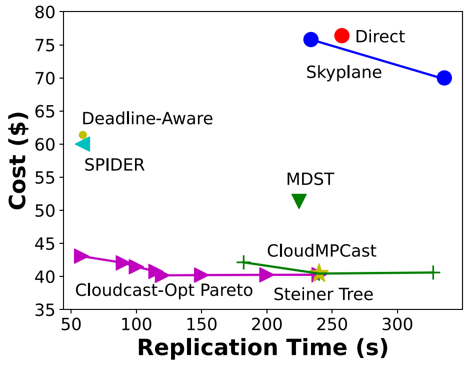
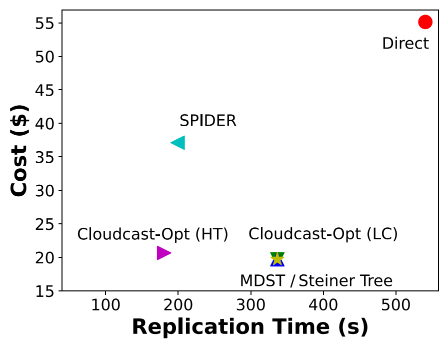


Figure 8: Simulated results for Multicast Algorithms.

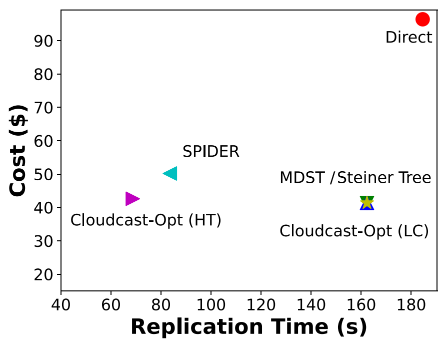
Cloudcast parameter settings through simulation. While we tested many configurations through the development of Cloudcast, due to limited space, we present results for a representative configuration[1]. Evaluated systems include Cloudcast-Opt, direct transmission to the destinations, sending along cost-minimizing trees (MDST and Steiner Tree), SPIDER [23], CloudMPCast [24], Skyplane [28], and a deadline-aware inter-DC optimizer [30]. Although Skyplane's optimizer is designed for unicast, not multicast, we adapt the optimizer's solution to multicast by running the optimization for each source-destination pair, and then combining all the graphs to build the distribution tree.

For Skyplane, CloudMPCast, and Cloudcast-Opt, we vary the throughput parameter to evaluate the performance range. For CloudMPCast [24], the optimizer allows for the level of throughput degradation to be controlled by an $\alpha \leq 1$ term, which determines how aggressively edges are filtered out. Our parameter sweep includes $\alpha \in [1, 0.5, 0.1]$, where $\alpha = 1$ maximizes CloudMPCast's throughput. For Skyplane, we vary the target throughput to maximize throughput and minimize cost, and plot both of these points. For Cloudcast-Opt, we show

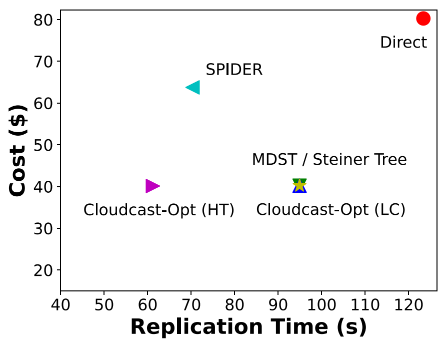
[1]Simulated Inter-Cloud: from `gcp:asia-southeast1-a` to `azure:eastasia`, `aws:af-south-1`, `azure:brazilsouth`, `aws:sa-east-1`



(a) AWS Intra-Cloud



(b) Azure Intra-Cloud



(c) GCP Intra-Cloud

Figure 9: Intra-cloud multicast results for algorithms implemented on Cloudcast.

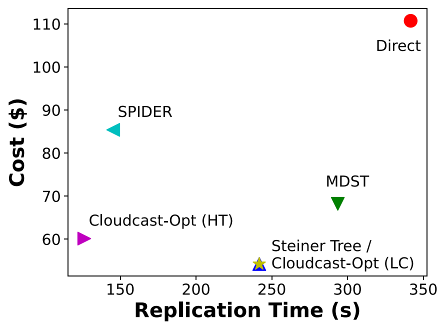


Figure 10: Inter-cloud multicast results for different algorithms implemented on Cloudcast. The Cloudcast replication tree is visualized in Figure 6.

results for several replication time constraints.

In Figure 8, we see that all baselines improve significantly upon direct transmission, and while some can match Cloudcast-Opt's capacity for fast replication time or low cost, no existing baseline can optimize both metrics simultaneously. Rather, Cloudcast-Opt's Pareto-curve can match or beat all baselines on at least one of cost or performance. CloudMP-Cast, whose α parameter does provide some flexibility, still offers a worse tradeoff than Cloudcast-Opt. Skyplane also has a significantly worse tradeoff curve, as it is not designed for multicast, so does not perform optimizations to alleviate source bottlenecks which are crucial for achieving high throughput. Despite this, even Skyplane's can improve throughput (for the throughput-maximizing solution) and reduce cost (for the cost-minimizing solution) as compared to direct transfers.

Cloud deployments. The remainder of our evaluations present empirical results from real cloud data transfers. Due to the high cost of running data multicast in the cloud ($20–$110 per transfer), we limit our evaluation to four representative configurations and four representative baselines identified by our simulation results. Among the configurations, three are intra-cloud replications corresponding to AWS[2], Azure[3] and GCP[4], and one is an inter-cloud replication workload that covers all three major providers[5]. These configurations are chosen to contain a source region with high egress costs to demonstrate potential cost savings. Among the baselines, we sub-selected the best-performing baselines from our simulation results in terms of throughput (SPIDER) and cost (Steiner Tree), with direct transmission providing a naive baseline.

Figures 9a to 9c show results for AWS, GCP, and Azure intra-cloud replication, and Figure 10 shows inter-cloud results. Across all configurations, given a very tight replication time constraint, Cloudcast-Opt (HT) solution leads to $46-62.4\%$ cost reductions and $2-2.84\times$ replication time speedup compared to sending directly to each destination.

Of the baselines tested, SPIDER [23] consistently demonstrates the lowest replication time, as it did in simulation. However, as SPIDER is not cost-aware, Cloudcast-Opt (HT) can achieve $28.4-44.0\%$ cost savings. Surprisingly, while saving significant cost, Cloudcast-Opt (HT) simultaneously speeds up replication by $1.11-1.35\times$, beating SPIDER on both axes. If, on the other hand, Cloudcast is given a loose replication time budget, i.e., Cloudcast-Opt (LC), it can find the cost-optimal solution in all setups, matching Steiner Tree solutions.

5.2 Cloud Provider and P2P Systems

We run end-to-end evaluation comparing Cloudcast with a commercial baseline (AWS S3 multi-region bucket replication) and P2P systems (BitTorrent and Bullet).

5.2.1 AWS S3 Multi-Region bucket replication

We run an end-to-end comparison between Cloudcast and AWS's S3 multi-region bucket replication [9] for single-provider multicast. AWS supports adding multiple replication rules to a source bucket to specify automatic replication to

[2]AWS Intra-Cloud: from `ap-east-1` to `us-west-1`, `ap-northeast-3`, `eu-north-1`, `ap-south-1`, `ca-central-1`, `ap-northeast-1`

[3]Azure Intra-Cloud: from `brazilsouth` to `westeurope`, `westus`, `koreacentral`, `australiaeast`, `uaenorth`, `centralindia`

[4]GCP Intra-Cloud: from `asia-southeast2-a` to `australia-southeast1-a`, `southamerica-east1-a`, `europe-west4-a`, `europe-west6-a`, `asia-east1-a`, `europe-west2-a`

[5]Inter-Cloud: from `gcp:asia-southeast1-a` to `azure:australiaeast`, `azure:eastasia`, `aws:ap-southeast-2`, `azure:brazilsouth`, `aws:sa-east-1`, `gcp:australia-southeast1-a`

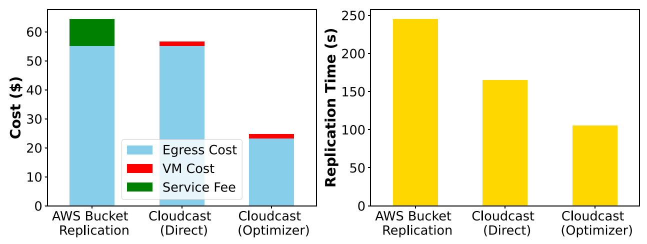


Figure 11: Cloudcast outperforms AWS S3 Replication Time Control while reducing total transfer costs.

one or more replication buckets. In the aspect of time control, AWS supports a replication time control with a minimum 15-minute SLO. However, we found that in our experiments, replications typically completed much faster than 15 minutes. Therefore, we use the actual replication time as a point of comparison.

We compare AWS's replication time and cost to Cloudcast with the planner implemented with both direct transfer and the optimizer. We transfer an OPT model [53] with 66 billion parameters (122 GB in total across 9 files) between regions in a single continent[6]. To evaluate AWS replication time and cost, we create buckets with replication rules from a bucket in the source region to buckets in destination regions. Once the replication rules are created, we copy data from a bucket in the same region into the source bucket with 16 VMs. After the write completes, we measure the time until the completion of replication into all destination buckets. We calculate the transfer cost according to AWS's pricing page [39]. We compare AWS multi-region bucket replication to Cloudcast implemented with both the direct and optimizer planner and running. As shown in Figure 11, the direct transfer has the same egress costs as AWS bucket replication, but the VM costs are much less than the service fee charged by AWS for the replication. Overall, Cloudcast with the optimizer is able to achieve 2.3× replication speedup and 61.5% cost savings. This is a result of being able to leverage VM parallelism as well as an overlay network that minimizes total egress costs.

5.2.2 P2P BitTorrent and Bullet

We also compare Cloudcast against P2P systems like BitTorrent and Bullet. We run the same transfer benchmark in Azure in Figure 9b, sending 100GB within Azure to 6 destination regions. We host our own BitTorrent tracker and use aria2 [48] as a BitTorrent client. Since Bullet's implementation is not available, we evaluate Bullet by implementing Bullet's algorithm inside Cloudcast's planner. The result is shown in Figure 12: both BitTorrent and Bullet have lower egress costs than direct but higher than Cloudcast. BitTorrent is the slowest because most clients cannot utilize the full bandwidth. The clients are built for scenarios like background seeding and transfer off the critical path, rather than for bulk

[6]from `aws:ap-east-1` to `aws:ap-southeast-2`, `aws:ap-south-1`, `aws:ap-northeast-3`, `aws:ap-northeast-2`, `aws:ap-northeast-1`

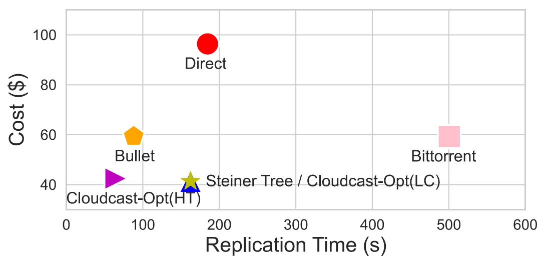
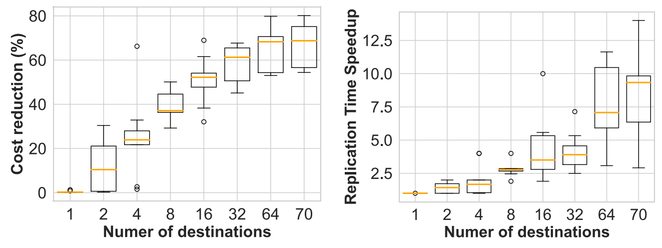


Figure 12: Comparison with BitTorrent protocol on the intra-cloud Azure workload in Figure 9b.



(a) Cost Reduction **(b)** Replication Time Speedup

Figure 13: Cloudcast optimizer's cost and time improvement over direct replication with varying destination numbers.

data transfer. Interestingly, without a centralized planner, BitTorrent is able to find a low-cost multicast replication tree by inferring the bandwidth among peers and preferring the data from peers who have the highest throughput. However, it is still significantly more expensive than Cloudcast.

5.3 Ablations of Cloudcast's Optimizer

To understand how our optimizer behaves for different selections of source and destination regions and different target replication times, we run simulated ablations.

5.3.1 Varying region selection

We test the generality of our improvements by randomly selecting source and destination regions for varying numbers of destinations. We show aggregated results over 100 samples for different numbers of destinations in Figure 13. Cloudcast is able to improve the runtime and cost of replication consistently across varying numbers of destinations. Cost and throughput improvement increase with more destinations, since more destinations provide a larger optimization space.

5.3.2 Impact of approximations on solutions

We evaluate how the optimizer with and without approximations scales to larger numbers of destinations in Figure 14, by randomly selecting source and destination regions for varying numbers of destination regions. We find that combining all three approximation mechanisms is necessary to scale the optimizer: using no approximations, or only one approximation, takes several minutes for just 10 destinations while using all approximations together reduces solve time to seconds.

We also evaluate how approximations affect the quality of the solution using the monetary cost of the solver-generated

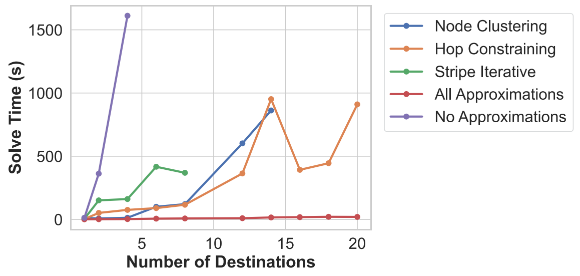


Figure 14: Approximations reduce solver runtime from the cutoff of 30 minutes to seconds for up to 20 destinations.

Method	Mean error	Solver speedup (geomean)
Node Clustering	0.3%	9.04×
Hop Constraining	1.1%	5.72×
Stripe Iterative	0.0%	7.02×
All Approximations	1.1%	30.68×

Table 3: Solve time and solution quality with approximations.

solution. We randomly sample 100 source/destination combinations for 5 destinations and compute the difference in the solution's monetary cost and replication runtime compared to MILP without approximation in Table 3. We find that the difference in cost averages around 1%, and estimate the worst-case approximation ratio to be 1.4. We find that for even just 5 destinations, the approximated solver runs with a geometric-mean speedup of 30.68×.

5.3.3 Accuracy of replication time model

We compare optimizer-modeled throughput and real throughput in Table 4. As transfer size increases, the approximation becomes more accurate. This is because Cloudcast's optimizer, designed for bulk data replication, makes several simplifying assumptions, such as perfectly pipelined stripes. Thus, transient inefficiencies during startup and teardown mean smaller transfers may experience lower throughput than the optimizer expects, but for larger, more expensive transfers, modeled throughput closely matches empirical results.

5.4 When to Use Cloudcast for Multicast?

Cloudcast is designed for bulk multicast replication in the cloud, so should only be used with data sizes are sufficiently large. Since Cloudcast relies on creating VMs in the cloud at transfer initiation time, there is a constant overhead from VM startup time. We calculate the transfer size break-even point (i.e. the minimum data size for using Cloudcast) for varying providers and VM capacity limits (constraining the throughput for the Cloudcast overlay), shown in Figure 15. We approximate the per-destination replication throughput without Cloudcast as equal to the per-VM egress bandwidth limit, ignoring congestion between source and destination VMs. Azure has a higher break-even point than AWS and GCP due to two effects. First, the VM startup time is the

Transfer Size (GB)	Prediction error
16	16.6%
32	8.51%
64	3.31%
128	1.69%

Table 4: Accuracy of the optimizer's predicted throughput.

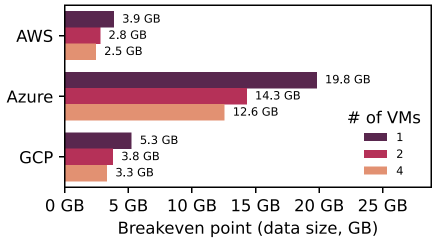


Figure 15: Estimated break-even point for a 6-destination replication based on VM startup times (35, 56, and 34 seconds for AWS, Azure, and GCP, respectively) and VM egress limits.

highest of all providers (56 seconds). Second, VMs in Azure are not subjected to egress constraints (5 Gbps and 7 Gbps for AWS and GCP, respectively). As a result, the benefits of using Cloudcast's techniques are only realized for larger transfer sizes or larger numbers of destinations.

6 Related Work

Overlay Unicast. A significant body of prior work uses overlay networks to improve the performance and resilience of one-to-one data transfers in the Internet and peer-to-peer networks [7, 12, 33]. In clouds, previous work has also leveraged cloud elasticity to further improve performance [28, 37]. However, they do not consider multicast, and except Skyplane [28], none consider the monetary cost of replication in the cloud. Handling multicast is challenging. For example, while [28] can leverage elastic resources, cloud pricing models, and overlay networking for bulk unicast replication in the cloud, its techniques are not directly applicable to the multicast setting. More specifically, Skyplane's flow-based throughput model results in ambiguous multicast distribution tree solutions as it ignores the identity of data sent along multiple paths. Furthermore, since Skyplane's optimizer is not designed for multicast, it cannot take advantage of techniques such as leveraging multiple distribution trees to alleviate source bottlenecks.

Overlay Multicast. End-system multicast [15] and overlay multicast have been proposed to efficiently disseminate data from a single source to multiple destinations. Many application-level multicast algorithms have been proposed. Algorithms like SPIDER [23], SplitStream [12], Bullet [33], and Overcast [29] are designed for high-bandwidth, cross-internet file distribution with application-level multicast overlays. However, these algorithms ignore monetary costs and focus on techniques to maximize bandwidth.

Inter-DC Replication. Extensive prior work addresses inter-DC replication [21, 22, 34, 44, 54], including bulk multicast [36]. Recent research includes deadline [36] and cost-awareness [20, 21]. However, the cost model in the Inter-DC setting cannot be easily adapted to cloud users, for which network pricing is based on total data volume rather than bandwidth (which introduces a non-linearity for a standard bandwidth-based MILP formulation). Furthermore, existing formulations are not designed for multicast [20, 21] or do not consider more than a few geo-distributed regions [36]. Our work focuses on public clouds, considering unique per-GB network pricing, elastic resources, and cloud-specific resource constraints. Our approximation algorithm is also designed to scale to all regions across multiple cloud vendors.

Traffic Engineering. The classic problem of traffic engineering has also formulated optimization problems for minimizing cost under performance constraints. These techniques have recently been applied to cloud providers and their monetary costs. For example, Entact [55] studied how to optimize costs for online service providers while still minimizing user latency. Similarly, Cascara [45] leveraged latency-equivalent paths to identify cost-minimizing paths for cloud providers. Like Inter-DC Replication, these approaches have been developed from the perspective of the cloud or service provider and, thus, present a materially different optimization problem.

Steiner Trees. The Steiner Tree algorithm has been applied in the multicast setting both to minimize costs in terms of delay [31] and cloud egress costs [24]). CloudMPCast [24] minimizes egress costs in cloud bulk data multicast by constructing a Steiner Tree overlay network that avoids low-throughput cross-region paths. However, CloudMPCast overlooks VM capacity and per-VM egress/ingress limits in its MILP formulation. Also, CloudMPCast aims to achieve *comparable* performance to direct transfers while minimizing cost, unlike Cloudcast, which optimizes throughput.

Geo-Distributed Storage. Geo-distributed storage via data replication is supported by a variety of cloud services, such as AWS Cross-Region Replication [9], AWS Multi-Region Access Points [11], and GCP Multi-region buckets [25]. Cross-region replicated buckets (e.g., S3 replication rules) automatically replicate written data from a bucket in one region to one or more buckets in other regions. However, these services have limited support for cross-cloud data movement and do not minimize egress costs even for intra-cloud data movement. SPANStore [51] designs a system for geo-distributed storage across multiple cloud providers, and also optimizes egress costs of relaying data on PUT requests. However, its relay strategy is optimized for latency, not bandwidth.

Peer-to-peer Multicast. Peer-to-peer systems (P2P) support file sharing among a set of end-user clients. The BitTorrent protocol [17] reduces the network load on the source by allowing clients to upload and download data to each other. BitTorrent is widely used for data multicast in data center environments by Facebook [19] and Twitter [49]. Specialized systems for data multicast that use BitTorrent include Uber's Kraken [40] and Ant Group's Dragonfly [6]. These P2P systems have significant overhead as they are designed for adversarial settings where peers may be unreliable or fail. Moreover, P2P systems must scale to millions of destinations and therefore lack centralized control which prevents custom routing topologies. P2P systems may redundantly send data over expensive links due to a lack of cost awareness.

METHOD	MULTICAST	CLOUD PRICING	STRIPING	RESOURCE ELASTICITY
Unicast overlay networks				
RON [7]	×	×	✓	×
Skyplane [28]	×	✓	✓	✓
COMS [20]	×	×	×	∼
Peer-to-peer				
BitTorrent [17]	✓	×	✓	×
SplitStream [12]	✓	×	✓	×
Bullet [33]	✓	×	✓	×
Inter-DC overlay multicast				
SPIDER [23]	✓	×	✓	×
CodedBulk [47]	✓	×	✓	×
BDS [54]	✓	×	✓	×
Deadline-aware Inter-DC [30]	✓	×	×	✓
Cost optimized overlay networks				
SPANStore [51]	✓	✓	×	×
CloudMPCast [24]	✓	✓	×	×
Jetway [21]	×	✓	✓	×
Cloudcast (ours)	✓	✓	✓	✓

Table 5: Cloudcast builds on prior work by enabling multicast, optimizing cloud costs, and leveraging cloud resource elasticity and multiple distribution trees.

7 Conclusion

In this paper, we explored the problem of cost-optimized cloud multicast by introducing overlay networks of ephemeral VM waypoints that exploit path-specific cloud pricing to significantly reduce cost and improve throughput. We developed a MILP formulation of this problem and introduced approximations that make the solving time feasible for practical applications. Our evaluation against academic and commercial baselines demonstrated up to a 61.5% reduction in cost and a 2.3× improvement in runtime. Cloudcast has been released as part of the Skyplane open source project with pluggable planning algorithms to enable future research in this space.

Acknowledgments

This work was supported by gifts from Accenture, AMD, Anyscale, Google, IBM, Intel, Microsoft, Mohamed Bin Zayed University of Artificial Intelligence, Samsung SDS, SAP, Uber, and VMware. We also thank Asim Biswal for his contributions to the open source artifact, and Daniel Rothchild for providing feedback on writing.

References

[1] Blackblaze. https://www.backblaze.com/b2/cloud-storage-pricing.html. Accessed: 2022-12-08.

[2] Wasabi. https://wasabi.com/. Accessed: 2022-12-08.

[3] Cvxpy: A python library for convex optimization, 2021.

[4] Lz4 - extremely fast compression, 2023.

[5] Pynacl: Python binding to the libsodium library, 2023.

[6] Alibaba. Dragonfly. https://github.com/dragonflyoss/Dragonfly, 2018. Accessed on 12/15/2022.

[7] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 131–145, 2001.

[8] Amazon AWS. Ec2 on-demand instance pricing. https://aws.amazon.com/ec2/pricing/on-demand, 2023.

[9] Aws cross-region replication. https://docs.aws.amazon.com/AmazonS3/latest/userguide/replication.html.

[10] Microsoft Azure. Pricing - bandwidth. https://azure.microsoft.com/en-us/pricing/details/bandwidth/, 2023.

[11] Alex Casalboni. Amazon s3 multi-region access points. https://aws.amazon.com/s3/features/multi-region-access-points/.

[12] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: High-bandwidth multicast in cooperative environments. *ACM SIGOPS operating systems review*, 37(5):298–313, 2003.

[13] Sarah Chasins, Alvin Cheung, Natacha Crooks, Ali Ghodsi, Ken Goldberg, Joseph E Gonzalez, Joseph M Hellerstein, Michael I Jordan, Anthony D Joseph, Michael W Mahoney, et al. The sky above the clouds. *arXiv preprint arXiv:2205.07147*, 2022.

[14] Yang Chu, Sanjay Rao, Srinivasan Seshan, and Hui Zhang. Enabling conferencing applications on the internet using an overlay muilticast architecture. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 55–67, 2001.

[15] Yang-hua Chu, Sanjay G Rao, Srinivasan Seshan, and Hui Zhang. A case for end system multicast. *IEEE Journal on selected areas in communications*, 20(8):1456–1471, 2002.

[16] Google Cloud. All networking pricing. https://cloud.google.com/vpc/network-pricing, 2023.

[17] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72. Berkeley, CA, USA, 2003.

[18] Frank Denis. The sodium cryptography library, Jun 2013.

[19] Facebook uses bittorrent, and they love it. https://torrentfreak.com/facebook-uses-bittorrent-and-they-love-it-100625/. Accessed on 12/15/2022.

[20] Bita Fatemipour, Wei Shi, and Marc St-Hilaire. A cost-effective and multi-source-aware replica migration approach for geo-distributed data centers. In *2022 IEEE Cloud Summit*, pages 17–22. IEEE, 2022.

[21] Yuan Feng, Baochun Li, and Bo Li. Jetway: Minimizing costs on inter-datacenter video traffic. In *Proceedings of the 20th ACM international conference on Multimedia*, pages 259–268, 2012.

[22] Jason Flinn, Xianzheng Dou, Arushi Aggarwal, Alex Boyko, Francois Richard, Eric Sun, Wendy Tobagus, Nick Wolchko, and Fang Zhou. Owl: Scale and flexibility in distribution of hot content. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 1–15, 2022.

[23] Samrat Ganguly, Akhilesh Saxena, Sudeept Bhatnagar, Rauf Izmailov, and Suman Banerjee. Fast replication in content distribution overlays. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 4, pages 2246–2256. IEEE, 2005.

[24] José Luis García-Dorado and Sanjay G Rao. Cost-aware multi data-center bulk transfers in the cloud from a customer-side perspective. *IEEE Transactions on Cloud Computing*, 7(1):34–47, 2015.

[25] Gcp multi-region bucket. https://cloud.google.com/storage/docs/locations#location-mr. Accessed on 12/15/2022.

[26] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[27] Frank K Hwang and Dana S Richards. Steiner tree problems. *Networks*, 22(1):55–89, 1992.

[28] Paras Jain, Sam Kumar, Sarah Wooders, Shishir G Patil, Joseph E Gonzalez, and Ion Stoica. Skyplane: Optimizing transfer cost and throughput using cloud-aware overlays. *arXiv preprint arXiv:2210.07259*, 2022.

[29] John Jannotti, David K Gifford, Kirk L Johnson, M Frans Kaashoek, and James W O'Toole Jr. Overcast: Reliable multicasting with an overlay network. In *Fourth Symposium on Operating Systems Design and Implementation (OSDI 2000)*, 2000.

[30] Siqi Ji, Shuhao Liu, and Baochun Li. Deadline-aware scheduling and routing for inter-datacenter multicast transfers. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 124–133, 2018.

[31] Jehn-Ruey Jiang and Szu-Yuan Chen. Constructing multiple steiner trees for software-defined networking multicast. In *Proceedings of the 11th International Conference on Future Internet Technologies*, pages 1–6, 2016.

[32] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[33] Dejan Kostić, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 282–297, 2003.

[34] Nikolaos Laoutaris, Michael Sirivianos, Xiaoyuan Yang, and Pablo Rodriguez. Inter-datacenter bulk transfers with netstitcher. In *Proceedings of the ACM SIGCOMM 2011 Conference*, pages 74–85, 2011.

[35] Long Luo, Qixuan Jin, Jingzhao Xie, Gang Sun, and Hongfang Yu. Cost-efficient scheduling of multicast transfers with deadline guarantees across edge datacenters. *IEEE Transactions on Services Computing*, 2021.

[36] Long Luo, Yijing Kong, Mohammad Noormohammadpour, Zilong Ye, Gang Sun, Hongfang Yu, and Bo Li. Deadline-aware fast one-to-many bulk transfers over inter-datacenter networks. *IEEE Transactions on Cloud Computing*, 10(1):304–321, 2019.

[37] Miguel Matos, António Sousa, José Pereira, and Rui Oliveira. Clon: Overlay network for clouds. In *Proceedings of the Third Workshop on Dependable Distributed Data Management*, pages 14–17, 2009.

[38] Meeting compliance requirements using s3 replication time control. `https://docs.aws.amazon.com/AmazonS3/latest/userguide/replication-time-control.html`. Accessed on 12/15/2022.

[39] Overview of data transfer costs for common architectures. `https://aws.amazon.com/blogs/architecture/overview-of-data-transfer-costs-for-common-architectures/`. Accessed on 12/15/2022.

[40] P2p docker registry capable of distributing tbs of data in seconds. `https://github.com/uber/kraken`. Accessed on 12/15/2022.

[41] Simon Peter, Umar Javed, Qiao Zhang, Doug Woos, Thomas Anderson, and Arvind Krishnamurthy. One tunnel is (often) enough. *ACM SIGCOMM Computer Communication Review*, 44(4):99–110, 2014.

[42] Matthew Prince and Nitin Rao. Aws's egregious egress. `https://blog.cloudflare.com/aws-egregious-egress/`, 2021.

[43] Daniel Rehfeldt and Thorsten Koch. Implications, conflicts, and reductions for steiner trees. In Mohit Singh and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization - 22nd International Conference, IPCO 2021, Atlanta, GA, USA, May 19-21, 2021, Proceedings*, volume 12707 of *Lecture Notes in Computer Science*, pages 473–487. Springer, 2021.

[44] Chijun Sima, Yao Fu, Man-Kit Sit, Liyi Guo, Xuri Gong, Feng Lin, Junyu Wu, Yongsheng Li, Haidong Rong, Pierre-Louis Aublin, et al. Ekko: A {Large-Scale} deep learning recommender system with {Low-Latency} model update. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 821–839, 2022.

[45] Rachee Singh, Sharad Agarwal, Matt Calder, and Paramvir Bahl. Cost-effective cloud edge traffic engineering with cascara. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 201–216, 2021.

[46] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 73–86, 2002.

[47] Shih-Hao Tseng, Saksham Agarwal, Rachit Agarwal, Hitesh Ballani, and Ao Tang. Codedbulk: Inter-datacenter bulk transfers using network coding. In *NSDI*, pages 15–28, 2021.

[48] Tatsuhiro Tsujikawa and Nils Maier. aria2 - the ultra fast download utility. `https://github.com/aria2/aria2`, 2008.

[49] Twitter uses bittorrent for server deployment. https://torrentfreak.com/twitter-uses-bittorrent-for-server-deployment-100210/. Accessed on 12/15/2022.

[50] Marcia Villalba. Amazon s3 replication adds support for multiple destination buckets. https://aws.amazon.com/blogs/aws/new-amazon-s3-replication-adds-support-for-multiple-destination-buckets/, 2020.

[51] Zhe Wu, Michael Butkiewicz, Dorian Perkins, Ethan Katz-Bassett, and Harsha V Madhyastha. Spanstore: Cost-effective geo-replicated storage spanning bmultiple cloud services. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 292–308, 2013.

[52] Zongheng Yang, Zhanghao Wu, Michael Luo, Wei-Lin Chiang, Romil Bhardwaj, Woosuk Kwon, Siyuan Zhuang, Sifei Luan, Gautam Mittal, Scott Shenker, and Ion Stoica. SkyPilot: An Intercloud Broker for Sky Computing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI '23)*, April 2023.

[53] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.

[54] 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 *Proceedings of the Thirteenth EuroSys Conference*, pages 1–14, 2018.

[55] Zheng Zhang, Ming Zhang, Albert G Greenberg, Y Charlie Hu, Ratul Mahajan, and Blaine Christian. Optimizing cost and performance in online service provider networks. In *NSDI*, pages 33–48, 2010.

A Optimizer Parameters

A.1 Stripe Granularity

The number of stripes is a parameter in the optimizer that determines the unit of data size that the optimizer determines routes. Choosing too small a number of stripes (e.g., 1-4) can result in solution infeasibility, since an individual stripe may be too large to fit along any given link under a replication time constraint. We show the tradeoff in 16 between solution quality (dollar cost) and the number of stripes set of solving a 3-destination topology. Adding more stripes can increase the solver runtime unnecessarily. We use 8 – 16 stripes for experiments.

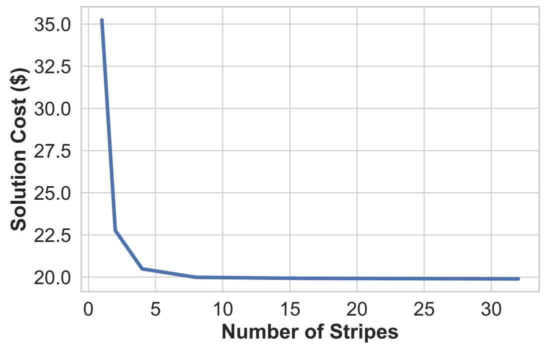


Figure 16: Solution quality v.s the number of stripes. Given a 6-destination intra-AWS transfer job and the same runtime SLO, our optimizer generates different solutions for different numbers of stripes. Small numbers of stripes can result in no feasible solution or solution with worse quality (i.e., higher cost). However, increasing the number of stripes to larger than 10 has diminishing returns.

A.2 Node Sub-Selection

In Figure 3.4, we describe how we cluster nodes to select a subset of nodes for consideration by the optimizer. We motivate this by running an experiment to randomly select a subset of nodes in Figure 17. Generally, there are diminishing returns (beyond 20 nodes) to consider additional nodes. To avoid variability from randomness, we use the techniques described in Figure 17 to select a represented subset of nodes.

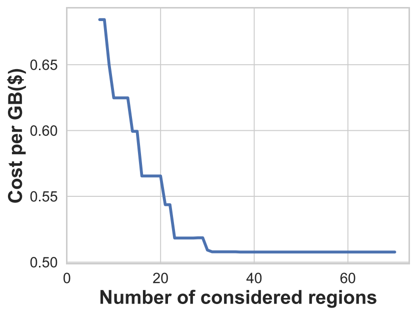


Figure 17: Solution quality versus the number of considered nodes. Considering a larger set of regions has diminishing returns of solution quality but exponentially increases optimizer runtime.

B Formulation Details

B.1 Ensuring Valid Paths

We cannot ensure connectivity of prevent cycles in paths defined by P without adding an exponential number of constraints. We define a special flow variable F to add constraints that ensure that flow can be pushed along paths from P from the source to all destinations. The source node is denoted as the source region index in the transfer, the destination nodes are denoted as the destination region indices, and the sink node is denoted as a special node that is only connected to destination nodes. We constrain F to have flow if and only if the corresponding stripe and edge for P is set to 1.

$$F_{s,(u,v)} \geq 1, \text{ if } P_{s,(u,v)} = 1 \tag{9}$$

We ensure zero or negative flow for $P_{s,u,v} = 0$ via capacity constraints. We set special capacity constraints between destination nodes and the sink, to ensure that the sink can only receive sufficient flow if it receives flow from all destinations.

$$F_{s,(u,v)} \leq \begin{cases} 1, & \text{if } u \in \textsc{dest}, v = \text{sink} \\ 0, & \text{if } P_{s,(u,v)} = 0 \\ |\textsc{dest}|, & \text{otherwise} \end{cases} \tag{10}$$

We impose conservation of flow $\forall s$:

$$\sum_{u \in V} F_{s,(u,v)} = \begin{cases} |\textsc{dest}|, & \text{if } v \text{ is the source} \\ -|\textsc{dest}|, & \text{if } v \text{ is the sink} \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

If the above constraints are met, this ensures that the stripe paths assigned by P are able to push flow from the source to all destinations.

B.1.1 Full Formulation

We can write a full formulation of an integer linear program as the following:

$$\arg\min_{P,N,F} \textsc{time} * \langle \textsc{cost}^{VM}, N \rangle + \sum_s \langle \textsc{cost}^{path}, P \rangle \tag{12}$$

$$N \leq \textsc{limit}^{VM} \tag{13}$$

$$\textsc{size}_{\textsc{stripe}} * \sum_s P_{s,(u,v)} \leq \textsc{capacity}^{path}{}_{u,v} \tag{14}$$

$$\textsc{size}_{\textsc{stripe}} * \sum_s \sum_{u \in V} P_{s,(v,u)} \leq \textsc{egress}^{VM} \tag{15}$$

$$\textsc{size}_{\textsc{stripe}} * \sum_s \sum_{v \in V} P_{s,(v,u)} \leq \textsc{ingress}^{VM} \tag{16}$$

$$F_{s,(u,v)} \geq 1, \text{ if } P_{s,(u,v)} = 1 \tag{17}$$

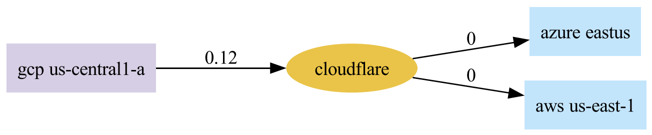
$$\sum_{u \in V} F_{s,(u,v)} = \begin{cases} |\textsc{dest}|, & \text{if } v \text{ is the source} \\ -|\textsc{dest}|, & \text{if } v \text{ is the sink} \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

$$F_{s,(u,v)} \leq \begin{cases} 1, & \text{if } u \in \textsc{dest}, v = \text{sink} \\ 0, & \text{if } P_{s,(u,v)} = 0 \\ |\textsc{dest}|, & \text{otherwise} \end{cases} \tag{19}$$

C How does Cheaper Egress Affect Cloudcast's Optimizations?

Some existing cloud providers (e.g., Cloudflare [42], Wasabi [2], and Blackblaze [1]) offer discounted or even free network egress. Interestingly, incorporating free-egress clouds into Cloudcast offers further opportunities to reduce costs. Figure 18a illustrates this effect. This highlights the importance of using a system like Cloudcast, which can adapt replication plans in response to cheaper network offerings.

It is possible that major cloud providers will also adapt free-egress models or, in a less extreme case, make intra-cloud network fees more uniform as they build up additional capacity for inter-region networks with limited bandwidth. In this case, the techniques used by Cloudcast (overlay networking, VM parallelism, and striping) would achieve only substantial throughput improvements but no cost improvement.



(a) Example of routing via no egress fee clouds.

	Direct	Cloudcast w/o Cloudflare	Cloudcast w/ Cloudflare
Cost ($/GB)	0.24	0.2075	0.12

(b) Egress cost is 2× cheaper including Cloudflare in Cloudcast.

Figure 18: Routing data through clouds with no egress fees (e.g., Cloudflare) can reduce inter-cloud replication costs as egress fees need to be paid only *once* rather than a minimum of *twice* to replicate data across AWS, GCP, and Azure.