CloudPilot: Flow Acceleration in the Cloud

Kfir Toledo[1,2], David Breitgand[2], Dean Lorenz[2], Isaac Keslassy[1]
[1] *Technion* [2] *IBM Research Haifa*

***Abstract*— TCP-split proxies have been previously studied as an efficient mechanism to improve the rate of connections with large round trip times. These works focused on improving a single flow. In this paper, we investigate how strategically deploying TCP-split proxies in the cloud can improve the performance of geo-distributed applications entailing multiple flows interconnecting globally-distributed sources and destinations using different communication patterns, and being subject to budget limitations.**

We present *CloudPilot*, a Kubernetes-based system that measures communication parameters across different cloud regions, and uses these measurements to deploy cloud proxies in optimized locations on multiple cloud providers. To this end, we model cloud proxy acceleration and define a novel *cloud-proxy placement problem.* Since this problem is NP-Hard, we suggest a few efficient heuristics to solve it. Finally, we find that our cloud-proxy optimization can improve performance by an average of $3.6\times$ for four different use cases.

I. Introduction

Motivation. Over the last few years, the fearsome competition among cloud providers has led them to spend billions on expanding their global presence by building data-centers worldwide and laying out high-speed lines to interconnect them [1]–[3]. Clients can now build on-demand cloud overlay networks comprising cloud nodes in different regions to route application traffic through the cloud rather than through the public internet [4]–[11].

Several studies [7]–[11] show how we can increase the rate of a flow by using cloud-based TCP-split proxies. As Fig. 1 shows, this method splits a single TCP flow into several connections with shorter round trip time (RTT). By reducing the RTT for each connection, the overall transmission rate is improved.

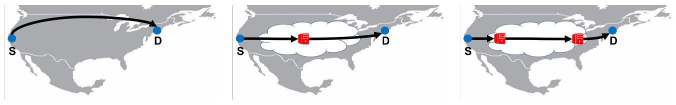
While the above works focus on accelerating single flows, it is unclear how to strategically deploy a limited set of cloud-based TCP-split proxies to improve the performance of global geo-distributed applications, with sources and destinations that need to exchange large amounts of data. Such applications include distributed databases, batch file exchanges, VM migrations, and CDNs [12]–[14] (§III). The goal of this paper is to introduce CloudPilot, a Kubernetes-based system that is designed to optimize proxy placement and deploy the proxies to serve these applications.

Contributions. We make the following contributions.

Use cases. We start by showing how geo-distributed applications can be modeled using four topological use-cases (§III).

CloudPilot. We develop and deploy CloudPilot, a Kubernetes-

ISBN 978-3-903176-48-5 ©2022 IFIP



(a) Direct connection **(b)** One-proxy acceleration **(c)** Two-proxy acceleration

Fig. 1: Connection types between host and destination.

based system that helps accelerate geo-distributed application traffic. Using communication parameters measured across different cloud regions, CloudPilot deploys TCP-split cloud proxies across multiple cloud providers to optimize the application transfer performance. We later present our proxy acceleration model for a single flow, and validate it using real-world CloudPilot-based measurement experiments (§IV). The open-source CloudPilot code is available online [15].

Proxy placement optimization. We explain how a natural metric of performance in geo-distributed applications is the *total FCT (Flow Completion Time)*, which we define to be the total time required to complete all necessary data transfers in a round of computation. We formally define a *cloud-proxy placement problem* to optimize this total FCT and prove that it is NP-hard (§V). Hence, we propose two families of heuristics algorithms to solve the problem: the *flow-greedy* algorithms, which greedily consider first the flows whose performance can most improve; and the *proxy-greedy* algorithms, which greedily establish first the proxies that can most improve performance (§VI).

Evaluations. We evaluate our proposed heuristics both through extensive simulations with parameters measured from actual cloud providers, and through real-world CloudPilot-based cloud-proxy deployments. We find that our heuristic algorithms achieve significant FCT acceleration. For example, spending 50¢-per-flow to transfer 2GB-flows on Google cloud decreases the total FCT by factors of 2.7, 3.6, 3.9 and 4.3 for four different application use-cases. We also find that counter-intuitively, FCT acceleration significantly improves as last-mile bandwidth increases, especially beyond 100Mbps, heralding an increased impact for CloudPilot with the last-mile fiber-optics deployment (§VII).

II. Related Work

TCP splitting. [16]–[18] show that TCP splitting can improve rate in different environments, such as mobile and satellite.

Cloud overlay network. Several works show that forwarding using cloud proxies without TCP splitting capability provides little to no improvement [4], [9], [10]. Later research shows the benefit of using TCP splitting in the cloud overlay network. [7]–[9] show that using a single TCP splitting proxy can achieve up to 3× improvement over a direct internet connection. [9] also suggests using Multi Path-TCP (MP-TCP) to increase performance. However, MP-TCP is not always supported by communicating parties. [10], [11] show we can achieve better performance by using two TCP-splitting proxies with large buffers, one close to the source and one to the destination. In addition, they implement several improvements for the TCP splitting, like TCP turbo start, which can also be implemented in our system and further increase its performance. However, they only consider isolated flows and not a full system. Several recent works also analyze in more detail the performance of cloud communications [1]–[3], [19].

Geo-distributed applications. Most research on the performance improvement of geo-distributed applications focuses on load-balancing mechanisms over direct TCP connections [20]–[24]. Our work complements these efforts by introducing TCP splitting, obtaining further significant performance gains.

Caching proxy placement. Several papers study the placement of cache proxies [25], [26] and HTTP-gathering proxies [5]. The TCP-split proxy placement problem is different. For example, in the above examples, it is preferable to place a cache as close to end-points as possible, while in TCP splitting, the preferred location of a single proxy is in the middle.

III. Use-cases

We are interested in considering many data-intensive geo-distributed applications. Since they may vary considerably, to reason about them, we abstract away details and focus on their characteristic communication patterns, classifying them into four topological use-cases.

One-to-many. A single source broadcasts information to many destinations worldwide. Usually, a Content Delivery Network (CDN) will be used for most of the destinations [14]. However, some 1.8% of all live streams use direct connections due to cache misses (Facebook statistics [27]). These remaining flows can be modeled using a star-like one-to-many pattern.

Many-to-many. Many nodes in different locations communicate in a full-mesh pattern, *e.g.,* in a geo-distributed database that transfers data between nodes to keep consistency [12].

Many one-to-one. Topology with many unrelated source-destination pairs. One example is a VM migration application [13] that entails sending data from one data-center location to another for many unrelated VMs. Additional examples include backup between data centers, and file-sharing systems.

Many localized one-to-many. Several sources that are geographically distributed and each broadcasts to many mostly-local destinations. One example is the traffic between CDN caches and their end-users [14]. Another is Twitch, an interactive live-streaming platform that offers three servers in three different continents [28].

In this paper, we focus on a static setting where applications have predictable traffic patterns (for instance, periodic backups), and we use this predictability to optimize the proxy placement by minimizing the total FCT of future flows. In future work, this could be extended to a speculative setting that uses various prediction models to estimate the upcoming future patterns. For example, an hourly process could consider the flow distribution in the last hour, then establish proxies for the next hour based on the expectation that the location distribution of future requests will be close enough. In addition, for simplicity, we focus in this paper on applications with at most dozens of flows per period.

IV. Proxy Acceleration System

In this section, we introduce our CloudPilot system that utilizes TCP-split proxies to reduce FCT. We derive a model to predict the FCT based on proxy properties and network measurements, and provide empirical evidence for the effectiveness of our approach. In the next section, we use our model to optimally decide which proxies to allocate for each flow.

A. CloudPilot

CloudPilot system. *CloudPilot* is a Kubernetes-based system that measures communication parameters across different cloud regions, and uses these measurements to deploy cloud proxies in optimized locations on multiple cloud providers. It is able to deploy new Kubernetes clusters on multiple cloud providers, deploy Kubernetes container instances, and connect between them. It also deploys iPerf3 measurement containers and HAProxy proxy containers. Finally, it implements the algorithms of this paper to decide where to deploy the proxies. The CloudPilot code is available online [15].

Deployment for FCT measurements. To obtain the FCT measurements below, CloudPilot spawns Kubernetes 1.22.2 clusters on demand for each pair of (source, destination) locations. The source cluster executes a Kubernetes container running an iPerf3 3.9 client [29] and the destination cluster executes a Kubernetes service with a backend container running an iPerf3 server. This way, FCT can be measured remotely between the source and destination over a direct cloud connection.

To obtain the FCT for connections with TCP-split proxies, CloudPilot creates additional Kubernetes clusters in different locations. CloudPilot can create three types of acceleration configurations: forwarding, one-proxy, and two-proxy.

For the forwarding acceleration, CloudPilot uses an Ubuntu 20.04 container. Traffic forwarding is done using appropriate iptables rules. For TCP splitting, CloudPilot uses an HAProxy 2.2.19 container that splits a TCP connection into two connections with smaller RTT. In addition, when CloudPilot uses two-proxy splitting acceleration, it increases the TCP buffer size of the containers to match their intra-cloud bandwidths between the proxies.

FCT measurements. Fig. 2 presents the results of our preliminary measurement experiments intended to gain an intuition about different flow proxy acceleration options. The rationale is to understand which options are likely to result in most gains and focus our exploration on those settings. Table I summarizes the configuration of each experiment. Fig. 2 shows how a forwarding proxy obtains either negligibly better or even worse performance compared to a direct communication over public

TABLE I: Configuration of CloudPilot measurement experiments

	source	dest.	1 proxy	2 proxies	
				first	second
a	Israel	California	London	London	Oregon
	public net	*public net*	*GCP*	*GCP*	*GCP*
b	London	Oregon	Montreal	London	Oregon
	AWS	*AWS*	*GCP*	*GCP*	*GCP*
c	Ohio	Mumbai	London	Virginia	Mumbai
	AWS	*AWS*	*GCP*	*GCP*	*GCP*
d	S. Carolina	Oregon	Iowa	S. Carolina	Oregon
	GCP	*GCP*	*GCP*	*GCP*	*GCP*

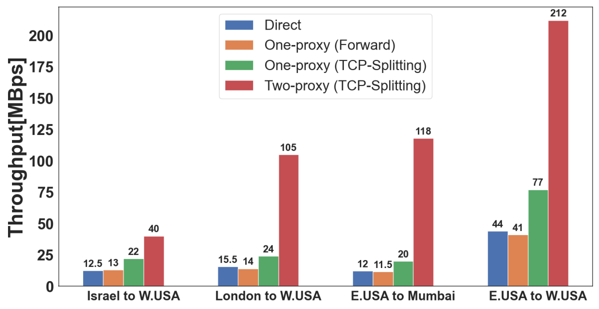


Fig. 2: Rate comparison between (1) direct connection, (2) one-proxy forwarding, (3) one-proxy splitting and (4) two-proxy splitting, using different types of (source, destination, proxies) tuples as described in Table I.

internet. This is consistent with previous studies [8], [10]. TCP-split proxies clearly outperform both direct connections and forwarding proxies. Therefore, in the remainder of this paper we focus on split proxies.

B. Proxy Acceleration Model

We now analyze how TCP-split proxies affect FCT and develop a model for estimating the FCT for several proxy deployment options: *direct connection*, *one proxy*, and *two proxies*, using measurable permanent properties of the end-hosts and the potential proxies. To do so, we consciously ignore the temporary impacts of the loss rate, queueing time, packet reordering, and similar effects along the packet path. In other words, we make the following simplifying assumptions: (1) each modeled flow transfers a large amount of data; (2) packet loss rate is negligible; (3) queueing time in the network is negligible vs. the propagation time; and (4) packet reordering is negligible. As we show in the next subsection, these assumptions are verified by our real cloud experiments.

Direct connection. We consider four main measurable factors affecting the FCT of a TCP flow from i to j.

Transfer size. Assume that i wants to transfer $\omega_{i,j}$ bits to j. Then FCT is directly proportional to the transfer size $\omega_{i,j}$ (using Assumption (1)).

Round Trip Time (RTT). The RTT equals $\textsc{Rtt}_{i,j}$, its propagation component between i and j (Assumption (3)).

Maximum window size. Let $\textsc{Wnd}_{i,j}$ be the maximum possible window size between i and j, as limited by the respective OS configurations. Since we send at most $\textsc{Wnd}_{i,j}$ bytes per RTT, the rate between i and j is bounded by $\frac{\textsc{Wnd}_{i,j}}{\textsc{Rtt}_{i,j}}$ [30].

Last-mile bandwidth. The flow's rate is limited by both the last-mile egress bandwidth of source i and the last-mile ingress bandwidth of destination j. The last-mile bandwidth may reflect a variety of factors, including the internet service provider rate limit or the NIC speed. We denote as $\mathrm{BW}_{i,j}$ the minimum of these two last-mile bandwidths.

Combining the above factors and applying assumptions, the flow rate is bounded by either $\frac{\textsc{Wnd}_{i,j}}{\textsc{Rtt}_{i,j}}$ or $\mathrm{BW}_{i,j}$, yielding an approximate rate of $R_{i,j} \approx \min\left(\frac{\textsc{Wnd}_{i,j}}{\textsc{Rtt}_{i,j}}, \mathrm{BW}_{i,j}\right)$. Its FCT $T^{\text{direct}}_{i,j}$ is approximated by $T^{\text{direct}}_{i,j} \approx \frac{\omega_{i,j}}{R_{i,j}}$.

One-proxy acceleration. FCT for a flow with one-proxy accelerator p is $T^{p}_{i,j} \approx \frac{\omega_{i,j}}{\min(R_{i,p}, R_{p,j})}$, because the flow rate is the rate of its slowest hop.

Two-proxy acceleration. FCT for a flow with two-proxy acceleration (p, q) is $T^{p,q}_{i,j} \approx \frac{\omega_{i,j}}{\min(R_{i,p}, R_{p,q}, R_{p,j})}$. As in [10], we assume that in the proxies, the maximum window sizes on the internet side use the Linux default. However, on the internal cloud side they can be increased to take full advantage of the paid cloud bandwidth rates, namely $R_{p,q} \approx \mathrm{BW}_{p,q}$, which is set by the cloud proxy capacity.

C. Model validation

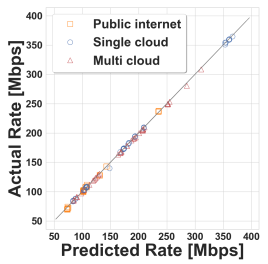
Fig. 3 puts our model to test in the real world. It plots the real-world measured *rate* and *rate acceleration* against the predicted values using our model, for all three types of connections. The figure shows sample results. Each data point averages 20 runs. Using CloudPilot we run three types of experiments: (1) *Public internet*, where source and destination are located in the public internet, and proxies are located in different regions of the same cloud, Google Cloud Platform (GCP) in this case. We use a desktop computer with Ubuntu v20.04 located at the Technion (Israel) as a host, and nine public iPerf3 servers around the world as destinations [29], [31]. (2) *Single cloud*, where the source, destination and proxies are sampled from 24 potential GCP locations. (3) *Multi cloud*, where source and destination are sampled from 9 IBM cloud locations, but the proxies are deployed in GCP. Overall we run over 75 different tuples (source, destination, proxies) with over $3,000$ tests. The flow average rate is measured for 40 seconds. Our model prediction is based on a maximum window size of 2.875MB (observed default for Linux TCP) for all links, except in the cloud-facing links of the proxies where they are set to 500MB. The proxy bandwidth limitation is estimated as $1750Mb/s$ (the per-flow limitation of HAProxy), as it is tighter than the $2Gb/s$ link capacity of our used proxy machines.

V. Cloud-proxy placement problem

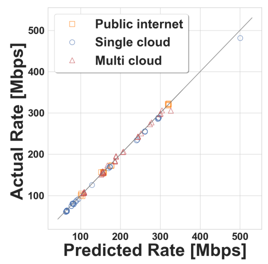
In this section, we present the cloud-proxy placement problem. First, we explain the problem informally to equip the reader with some intuition. Next, we introduce a formal notation and present a MILP (mixed-integer linear programming) formulation of the problem.

A. Informal Problem Definition

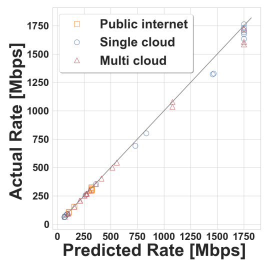
Given a set of source-destination pairs representing TCP flows, we want to find a feasible allocation of the flows to a set



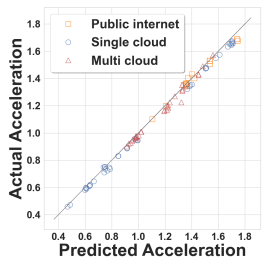
(a) Rate in direct connection



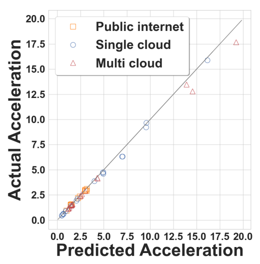
(b) Rate with one proxy



(c) Rate with two proxies



(d) One-proxy rate acceleration



(e) Two-proxy rate acceleration

Fig. 3: Model validation: Real-world *rate* vs. predicted rate using (a) direct connection; (b) one proxy; and (c) two proxies; then real-world *rate acceleration* (beyond direct connection) vs. predicted one using (d) one and (e) two proxies. In all cases, the model predictions seem close to real-world measured values.

Notation	Description
	Input Sets
$\mathcal{S}$	Set of all servers in the system, $s_i \in \mathcal{S}$
$\mathcal{F}$	Set of all valid flows in the system, $f_{i,j} \in \mathcal{F}$
$\mathcal{L}$	Set of all possible regions for proxies, $l \in \mathcal{L}$
$\mathcal{N}$	Set of all possible instances for proxy, $n \in \mathcal{N}$
$\mathcal{P}$	Set of all possible proxies $p \in \mathcal{P}$, $\mathcal{P}$ contains all instances in all locations, $\mathcal{P} \equiv (\mathcal{L} \times \mathcal{N})$
$\mathcal{A}$	Set of all possible proxy assignments (2 proxies, 1 proxy, or direct connection), $\mathcal{A} = (\mathcal{P}\times\mathcal{P}) \cup (\mathcal{P}\times\{0\}) \cup \{(0,0)\}$
	Input Parameters
$\omega_{i,j}$	Data size to transfer by $f_{i,j}$
$BW\,(p)$	Bandwidth capacity of proxy $p \in \mathcal{P}$
$C_{\text{BW}}\,(p)$	Network traffic cost per Gigabyte using proxy $p \in \mathcal{P}$
$C_{\text{setup}}\,(p)$	Cloud proxy setup cost for proxy $p \in \mathcal{P}$
B	Maximum allowed budget in the system
	Computed FCTs by CloudPilot (§IV)
$T^{\text{direct}}_{i,j}$	FCT of $f_{i,j}$ using direct path
$T^{p}_{i,j}$	FCT of $f_{i,j}$ using one proxy $p \in \mathcal{P}$
$T^{p,q}_{i,j}$	FCT of $f_{i,j}$ using two proxies $p, q \in \mathcal{P}$
	Decision Variables
$u^{p,q}_{i,j}$	$\begin{cases} 1 & \text{flow } f_{i,j} \text{ uses proxies } p,q \in \mathcal{P} \\ 0 & \text{otherwise} \end{cases}$
x_k	$\begin{cases} 1 & \text{if proxy } k \in \mathcal{P} \text{ is used} \\ 0 & \text{otherwise} \end{cases}$

Input

$$\mathbf{T}_{ij}[p,q] = \begin{cases} T^{\text{direct}}_{i,j} & \text{if } (p,q) = (0,0) \\ T^{p}_{i,j} & \text{if } (p,q) = (p,0) \\ T^{p,q}_{i,j} & \text{otherwise.} \end{cases} \quad \forall (p,q) \in \mathcal{A}$$

(FCTs from §IV)

Optimization goal

$$\text{minimize} \sum_{\forall f_{i,j} \in \mathcal{F}} \sum_{(p,q) \in \mathcal{A}} \mathbf{T}_{i,j}[p,q] \cdot u^{p,q}_{i,j} \qquad \text{(total FCT)}$$

Constraints

$$\sum_{(p,q) \in \mathcal{A}} u^{p,q}_{ij} = 1 \quad \forall f_{i,j} \in \mathcal{F} \qquad \text{(one allocation per flow)}$$

$$BW\,(k) \geq \sum_{\forall f_{i,j} \in \mathcal{F}} \omega_{i,j} \left(\sum_{(p,k) \in \mathcal{A}} \frac{u^{p,k}_{i,j}}{\mathbf{T}_{i,j}[p,k]} + \sum_{(k,q) \in \mathcal{A}} \frac{u^{k,q}_{i,j}}{\mathbf{T}_{i,j}[k,q]} \right) \quad \forall k \in \mathcal{P}$$

(proxy capacity fulfills bandwidth demand)

$$x_k \leq \sum_{f_{i,j} \in \mathcal{F}, (p,q) \in \mathcal{A} \text{ s.t. } p=k \vee q=k} u^{p,q}_{i,j} \quad \forall k \in \mathcal{P} \qquad \text{(0 if } k \text{ unneeded)}$$

$$|\mathcal{N}| x_k \geq \sum_{f_{i,j} \in \mathcal{F}, (p,q) \in \mathcal{A} \text{ s.t. } p=k \vee q=k} u^{p,q}_{i,j} \quad \forall k \in \mathcal{P} \qquad \text{(1 if } k \text{ needed)}$$

$$x_{k_1} = x_{(l,n_1)} \geq x_{(l,n_2)} = x_{k_2} \quad \forall n_1 < n_2 \in \mathcal{N}, \forall l \in \mathcal{L}$$

(for proxies k_1, k_2 with the same location l, prefer the smaller index)

$$C^{total}_{\text{setup}} = \sum_{k \in \mathcal{P}} C_{\text{setup}}\,(k)\, x_k \qquad \text{(total setup cost)}$$

$$C^{total}_{\text{BW}} = \sum_{f_{i,j} \in \mathcal{F}} \omega_{i,j} \sum_{p,q \in \mathcal{A}} u^{p,q}_{i,j} \left(C_{\text{BW}}\,(p) + C_{\text{BW}}\,(q) \right) \qquad \text{(total BW cost)}$$

$$B \geq C^{total}_{\text{setup}} + C^{total}_{\text{BW}} \qquad \text{(budget limitation)}$$

Fig. 4: MILP formulation of cloud-proxy placement problem, with a table of used notations on the left.

of TCP-split proxies in cloud regions, such that we minimize the total FCT (the sum of per-flow FCTs). An allocation is feasible if (a) its cost is no greater than the overall predefined budget, and (b) for any proxy, the sum of bandwidth demands of all flows using this proxy is no greater than its capacity. Each flow can be allocated one, two, or zero proxies (the latter corresponds to a direct connection).

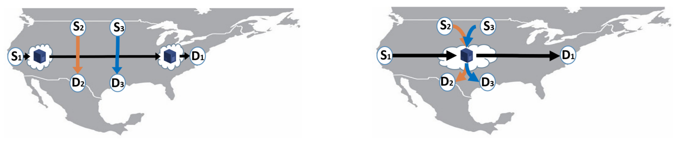
The cost of using a proxy comprises two components: (1) the proxy setup cost (*e.g.,* a virtual machine or a container with a specific bandwidth capacity), and (2) the data-transmission cost. If more than one flow share a proxy in a feasible allocation, the setup cost is paid only once. Note that cloud providers do not impose costs on the inbound network traffic. Only the outgoing traffic from the cloud proxy is billed (to different regions or to exit the cloud).

B. *Problem Statement*

Fig. 4 presents a formal MILP formulation for our cloud-proxy placement problem. Formally, we want to find a feasible assignment of proxies to flows such that the total FCT in the system is minimized, given constraints that reflect (1) the input sets and parameters presented on the top-left of Fig. 4, including the proxy setup costs, proxy data-transfer costs, and total budget; and (2) the per-flow FCT of each flow using any zero, one or two proxies, as computed by the proxy acceleration model of §IV.

The following theorem (proved in the Appendix) states that this problem is NP-hard.

Theorem 1. *The cloud-proxy placement problem is NP-Hard.*



(a) Flow-greedy prefer two proxies **(b)** Proxy-greedy prefer single proxy

Fig. 5: Intuition for algorithm choices. (a) Flow-greedy algorithms pick the best proxy acceleration for the flow that benefits most, even if expensive, and tend to prefer two-proxy acceleration; while (b) proxy-greedy algorithms choose the single proxy that can most benefit the system by serving several flows, thus spending the budget more efficiently.

VI. Algorithms

Since the cloud-proxy placement problem is NP-hard, we propose two families of greedy approximation algorithms: (1) the *flow-greedy* family of algorithms, where we greedily allocate flows, one at a time; and (2) the *proxy-greedy* family of algorithms, where we greedily allocate proxies, one at a time. Fig. 5 provides an example for understanding the intuition behind the two families of algorithms.

A. Flow-greedy algorithms

We propose two versions of the flow-greedy algorithm that differ only by the gain calculation, namely the order of processing flows.

F-FCT (Flow-greedy FCT). The pseudo-code for *F-FCT* is given in Alg. 1. It takes as input (Line 1) the set of flows, the set of proxies, and the overall budget. For each flow, *F-FCT* initializes the allocation to a direct connection (Line 4) and computes its gain for every possible proxy allocation (Line 5). That is, it considers all possible locations for one proxy or one proxy pair and computes the gain w.r.t. a direct connection. The possible allocations for each flow are sorted by their gain.

The main loop (Line 10) greedily processes flows one at a time, launching the greedy function that examines the best proxy locations for each flow, and updates the flow with the highest gain (Line 15). It then finds concrete proxy instances at these locations and calculates the cost of allocating these instances to the flow. If the budget allows, then the allocation for the flow is completed (Line 31) and the greedy step concludes. In order to find concrete proxy instances and their marginal cost (Line 18), Alg. *F-FCT* calculates the needed rate through the allocated proxies (Line 20). Then, it looks for a proxy with enough free capacity at each location (Line 24). It first tries to find an existing proxy with enough available capacity (Lines 25-26); if that fails, it uses a new proxy instance. The cost of the allocation (Line 22) includes the bandwidth cost of each proxy and the setup cost if a new proxy instance is required.

The gain function used by Alg. *F-FCT* (Line 13) only considers the reduction in FCT, regardless of its impact on the total budget. As illustrated in Fig. 5, F-FCT prefers expensive two-proxy acceleration types that strongly reduce the FCT, rather than cost-efficient one-proxy connections. Intuitively, F-FCT is best to use when the budget is nearly unlimited.

Time complexity. Let m be the number of flows, L the number of regions, and n the number of instances for each region.

Algorithm 1 Flow Greedy FCT (F-FCT)

```
 1: MAIN($\mathcal{F}^0, \mathcal{P}^0, B^0$)                 ▷ Flow and proxy sets, budget
 2:   $\mathcal{D} \leftarrow (\mathcal{L} \times \mathcal{L}) \cup (\mathcal{L} \times \{0\}) \cup (0,0)$   ▷ $\mathcal{D}$ is a list of every possible
                                                    proxy allocation location
 3:   for $f \in \mathcal{F}^0$ do
 4:     $a_f \leftarrow (0,0)$                   ▷ $a_f$ is the allocation for $f$, initialized as a direct connection
 5:     $\mathbf{G}_f \leftarrow \mathcal{D}$, sorted non-increasing by GAIN($f,d$) $\forall d \in \mathcal{D}$
                                          ▷ $\mathbf{G}_f$ is a list of all possible $\mathcal{A}$
                                            for $f$ sorted by gain
 6:   $r_p \leftarrow BW(p) \quad \forall p \in \mathcal{P}^0$           ▷ Available proxy bandwidth
 7:   $\mathcal{P} \leftarrow \emptyset$                         ▷ Allocated proxies so far
 8:   $B \leftarrow B^0$                            ▷ Remaining budget
  — end of initialization —
 9:   $\mathcal{F} \leftarrow \mathcal{F}^0$                     ▷ Flows without allocated proxies
10:   while $\mathcal{F} \neq \emptyset$ do GREEDY-STEP
11:   return $\sum_{f \in \mathcal{F}^0} \mathbf{T}_f[a_f], \{a_f\}_{f \in \mathcal{F}^0}$   ▷ Return overall score and allocation per flow

12: GAIN($f, a$)          ▷ FCT reduction for $f$ with allocation $a = (p, q)$
13:   return $\mathbf{T}_f[0,0] - \mathbf{T}_f[a]$

14: GREEDY-STEP
15:   $f \leftarrow \arg\max_{f \in \mathcal{F}} (\text{GAIN}(f, \mathbf{G}_f.head))$   ▷ Greedily choose flow
16:   $a, b \leftarrow$ FIND-PROXY-INSTANCES($f$)
17:   if $b \leq B$ then ALLOCATE($f, a, b$)

18: FIND-PROXY-INSTANCES($f$)
19:   $(l_1, l_2) \leftarrow \mathbf{G}_f.head$
20:   $r \leftarrow \omega_f / \mathbf{T}_f[l_1, l_2]$                 ▷ Flow BW requirement
21:   $a \leftarrow$ (FIND-PROXY-AT($l_1, r$), FIND-PROXY-AT($l_2, r$))
22:   $b \leftarrow \sum_{p \in a, p \neq 0} \omega_f C_{\text{BW}}(p) + \sum_{p \in a, p \notin \mathcal{P}} C_{\text{setup}}(p)$   ▷ Marginal allocation cost
23:   return $a, b$              ▷ Return chosen proxy instances, $a$, and their marginal cost, $b$

24: FIND-PROXY-AT($l, r$)   ▷ Find proxy at location $l$ with $r$ free capacity
25:   $\mathcal{P}_l \leftarrow \{p \in \mathcal{P}$ s.t. $p$'s location is $l\}$      ▷ $\mathcal{P}_l \leftarrow \emptyset$ if $l = 0$
26:   if $\exists p \in \mathcal{P}_l$ s.t. $r_p \geq r$ then
27:     return $p$    ▷ Proxy instance $p$ has enough capacity for the flow
28:   if $\exists p \in \mathcal{P}^0 \setminus \mathcal{P}$ s.t. $p$'s location is $l$ then   ▷ Always False if $l = 0$
29:     return $p$                                ▷ New proxy instance
30:   return 0

31: ALLOCATE($f, a, b$)                  ▷ Allocate $a$ to $f$ with budget $b$
32:   $a_f \leftarrow a, \quad B \leftarrow B - b, \quad \mathcal{F} \leftarrow \mathcal{F} \setminus \{f\}$
33:   for $p \in a$ do
34:     $\mathcal{P} \leftarrow \mathcal{P} \cup \{p\}$                        ▷ Add proxy if new
35:     $r_p \leftarrow r_p - \omega_f / \mathbf{T}_f[a]$            ▷ Update available capacity
```

The size of $\mathcal{D}$ is $O(L^2)$, so sorting for all flows requires $O(mL^2 \log(L))$ time. Each greedy step requires $O(m+n)$ time, $O(m)$ to find the best flow[1] and $O(n)$ to find its concrete proxy allocation. One flow is removed after each successful greedy step, thus the total time for the successful steps is $O(m(m+n)) = O(m^2)$ (since $n \leq m$). In order to bound the work required to process unsuccessful greedy steps, after each successful allocation, we make sure that the best potential allocation for each flow (the head of its sorted list) falls within the remaining budget. This is done by removing infeasible allocations from the head of each flow's list (in $O(1)$ per removal).[2] There are at most $O(mL^2)$ such removals, so overall $O(mL^2)$ time is required. Summing all, we get $O(m(L^2 \log(L) + m))$ time for Alg. *F-FCT*.

F-Cost (Flow-greedy FCT per cost). This algorithm is similar

[1]Note that the gain for each allocation can be cached.

[2]This implementation detail is omitted from Alg. 1 to simplify the presentation. The check can be done in $O(1)$ by caching $\max_{p \in \mathcal{P}_l} r_p$ (Line 26).

Algorithm 2 Flow Greedy Cost (F-Cost) *extends* Alg. 1

```
1:  GAIN(f, a)
2:      return $\frac{\mathbf{T}_f[0,0]-\mathbf{T}_f[a]}{\sum_{p\in a}\left(\omega_f C_{\text{BW}}(p)+C_{\text{setup}}(p)\frac{\omega_f/\mathbf{T}_f[a]}{BW(p)}\right)}$   ▷ $\frac{\omega_f}{\mathbf{T}_f[a]}$ is $f$'s rate
```

Algorithm 3 One-Proxy Greedy (1-P)

$\mathcal{Q}^{\mathcal{F}} = \{a_f\}_{f\in\mathcal{F}}$ denotes a set of proxy allocations a_f for all flows f

```
1:  MAIN
2:      $Score_{min}, \mathcal{Q}^{\mathcal{F}}_{min}, \mathcal{P}_{min} \leftarrow \infty, \{(0,0)\}_{f\in\mathcal{F}}, \emptyset$   ▷ Initialization
3:      do
4:          $update \leftarrow False$   ▷ Flag indicates score improvement
5:          for each $\mathcal{P}$ in CANDIDATE-PROXY-SETS($\mathcal{P}_{min}$) do
6:              $Score, \mathcal{Q}^{\mathcal{F}} \leftarrow$ FLOW-GREEDY-FCT($\mathcal{F}, \mathcal{P}_l, B$)
7:              if $Score < Score_{min}$ then
8:                  $Score_{min}, \mathcal{Q}^{\mathcal{F}}_{min}, \mathcal{P}_{min} \leftarrow Score, \mathcal{Q}^{\mathcal{F}}, \mathcal{P}_l$
9:                  $update \leftarrow$ True
10:     while $update$   ▷ Stop if no candidate improved score
11:     return $Score_{min}, \mathcal{Q}^{\mathcal{F}}_{min}$

12: CANDIDATE-PROXY-SETS($\mathcal{P}$)
13:     return ADD-ONE-PROXY($\mathcal{P}$)

14: ADD-ONE-PROXY($\mathcal{P}$)
15:     for each region $l \in \mathcal{L}$ do
16:         choose a proxy $p_l$ in region $l$ s.t. $p_l \notin \mathcal{P}$
17:         $\mathcal{P}_l \leftarrow \mathcal{P} \cup \{p_l\}$
18:     return $\{\mathcal{P}_l\}_{l\in\mathcal{L}}$
```

to the previous one, but considers cost when greedily choosing flows to process. The pseudo code is given in Alg. 2; it uses the same code of Alg. 1 with the GAIN function replaced. The idea is to scale down the gain for each allocation by its expected cost. The flow rate for each potential allocation can be computed at initialization from its expected FCT, so the BW cost of each allocation is known. However, the exact setup cost for each allocation cannot be known at initialization, since it depends on whether the allocation would use an existing proxy with enough free capacity or would require a new instance. Instead, Alg. 2 attributes a fraction of the setup cost for every allocated flow using the ratio of the flow rate to the capacity of the proxy. Note that this cost-based gain is only used to sort the flows and allocations and is *not* used to calculate the actual allocation cost (Alg. 1, Line 22). With this algorithm, we get better performance under a limited budget. The time complexity is the same as for Alg. 1.

B. Proxy-greedy algorithms

In the *proxy-greedy* family of algorithms, we choose the best proxy locations incrementally in a greedy manner. We start from an empty proxy set and add a few proxies at a time, so long as the overall FCT improves. At each greedy step, we generate a list of candidate proxy sets, and choose the one with the best total FCT. The total FCT for each candidate proxy set is computed using Alg. 1. The difference between the algorithms is in the way the candidate sets are generated at each greedy step.

1-P (one-proxy greedy). This is the basic proxy-greedy algorithm, its pseudo-code is given in Alg. 3. Alg. 3 creates a candidate set that includes all possibilities of adding a single

Algorithm 4 Two-Proxy Greedy (2-P) *extends* Alg. 3

```
1:  CANDIDATE-PROXY-SETS($\mathcal{P}$)
2:      return ADD-TWO-PROXIES($\mathcal{P}$)

3:  ADD-TWO-PROXIES($\mathcal{P}$)
4:      $\mathbb{P}^2 \leftarrow \emptyset$
5:      $\mathbb{P}^1 \leftarrow$ ADD-ONE-PROXY($\mathcal{P}$)
6:      for each combination $\mathcal{P}^l \in \mathbb{P}^1$ do
7:          append ADD-ONE-PROXY($\mathcal{P}^l$) to $\mathbb{P}^2$
8:      return $\mathbb{P}^2$
```

Algorithm 5 Two-Proxy Rollback (2-P RB) *extends* Alg. 3

```
1:  CANDIDATE-PROXY-SETS($\mathcal{P}$)
2:      $\mathbb{P}^{RB} \leftarrow \emptyset$
3:      for each $p \in \mathcal{P}$ do
4:          append ADD-TWO-PROXIES($\mathcal{P} \setminus p$) to $\mathbb{P}^{RB}$
5:      return $\mathbb{P}^{RB}$
```

proxy instance to the existing set (Lines 5&14). For every possible location in $\mathcal{L}$, it creates a candidate proxy set that includes the existing proxies plus a new proxy instance at that location. Then, at each greedy step, Alg. 1 is called for every proxy set in the candidate set to compute its FCT score (Line 6).[3] If a candidate has a better total FCT score then its allocation is saved. The algorithm returns if no candidate proxy set improves the total FCT (Line 10).

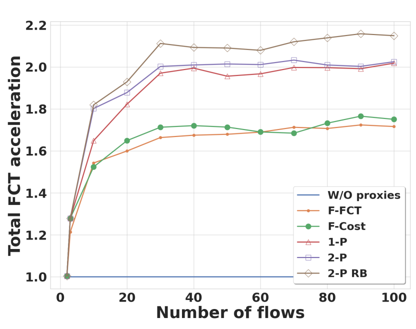
Time complexity The candidate set size is bounded by the number of locations and the number of greedy steps is $O(m)$, since there are at most 2 proxies per flow. Thus there are (mL) calls to Alg. 1. Each call requires $O(m(L^2\log(L)+m))$, however the initialization sorting can be cached to reduce subsequent calls to $O(m(L^2+m))$.[4] The overall time complexity is thus $O(m^2L(L^2+m))$. Let P denote the number of proxies returned by the algorithm. Both the number of greedy steps and the number of available locations for each is bounded by P. Thus there are (PL) calls to Alg. 1 each requiring $O(m(P^2+m))$. The overall complexity becomes $O(mPL(P^2+m))$, which is tighter in practice as P is limited by the overall budget.

2-P (two-proxy greedy). The algorithm is based on Alg. 3, but with a candidate set that now includes all possibilities of adding two-proxy instances to the existing set (Line 3). The implementation reuses ADD-ONE-PROXY to generate the candidate set. The candidate set size is now $O(L^2)$, therefore the overall time complexity increases to $O(m^2L^2(L^2+m))$ and $O(mPL^2(P^2+m))$.

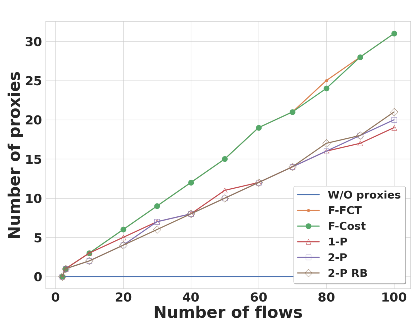
2-P RB (two-proxy greedy with rollback). The algorithm is again based on the Alg. 3, but with a candidate set that now includes all possibilities of *removing* one proxy and adding two proxy instances to the existing set (Line 1). The idea is to avoid local minima by allowing the greedy algorithm to rollback one of the existing proxy allocations when it adds new proxies. Here we reuse ADD-TWO-PROXY from Alg. 4.

[3] Note that the proxy set defines how many instances are available at each location, thus FIND-PROXY-AT may return 0 also for $l \neq 0$.

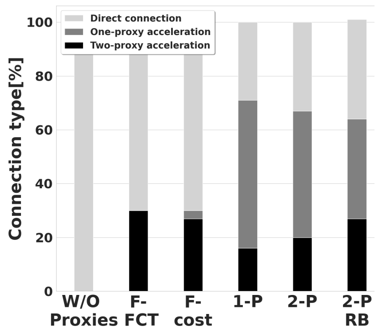
[4] In practice, the bound on L is smaller for most calls, as we only need to consider the locations that are covered by each particular candidate proxy set ($\{L$ s.t. $\mathcal{P}_l \neq \emptyset\}$).



(a) Total FCT acceleration



(b) Number of proxies



(c) Connection-type distribution

Fig. 6: Impact of number of flows in one-to-many use case. (a) shows the FCT acceleration when compared to a baseline without proxy. The proxy-greedy family of algorithms outperforms the flow-greedy algorithms and obtains over $2\times$ acceleration. (b) shows that the flow-greedy algorithms tend to spend a larger share of the budget on establishing proxies. (c) details each family's connection type distribution with 60 flows, confirming the intuition from Fig. 5 that flow-greedy algorithms tend to choose expensive single-use two-proxy accelerations for flows, while proxy-greedy algorithms prefer cheaper one-proxy accelerations with proxy sharing.

Now the size of the candidate set is $O(L^3)$, so the overall time complexity is $O(m^2L^3(L^2+m))$ or $O(mPL^3(P^2+m))$. Although the above theoretical complexity bound is high, we found the actual run-time to be acceptable in practice. Both the number of world-wide cloud geographic locations and the number of flows is relatively small (dozens). Due to budget constraints, the number of allocated proxies is even smaller.

VII. Evaluation

First, in simulations based on real-world parameters, we study the impact of several key model parameters and evaluate the performance of our algorithms on the use cases of §III. Then, in CloudPilot-based real-world cloud-environment experiments using Kubernetes and HAProxy, we confirm that the model predictions are close to reality, and that the proxy acceleration can be significant.

A. Settings

Runs. Each simulation data point is an average of 30 runs.

Proxy locations. We use 18 actual GCP regions for possible locations of the cloud proxies. The RTTs between the proxies are measured by CloudPilot and are consistent with a GCP RTT benchmark [32]. Due to lack of space, we present only the GCP results, but we get similar results in other cloud platforms that we checked, *e.g.,* IBM cloud.

Source and destination locations. To deploy each source, we first choose a random proxy, then select a location such that it has a reasonably small RTT to this proxy. We randomly choose locations with $RTT_{i,p} < 40$ms, corresponding to some 8,000Km using optical fibers [33]. Destination locations are chosen in the same way.

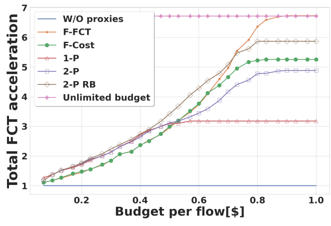
Network parameters. We set the transferred data size as $\omega = 2$GB for each flow. We use a default constant proxy setup cost of $C_{\text{setup}}(p) = 50$¢ and constant bandwidth cost of $C_{\text{BW}}(p) = 8$¢ per GB for all proxies and regions, approximating the GCP prices [34], [35]. We set the proxy bandwidth capacity to 2Gbits since this is a standard egress bandwidth of a container on GCP [36]. We set the last-mile bandwidth BW of all our end-hosts to be 1Gbps, planning for a next-generation widespread gigabit access, at least among corporate customers [37]; except for the multi-flow servers, such as in the CDN and one-to-many use cases, which are not constrained by last-mile bandwidth. We use the default Linux window size for all servers and for one-proxy connections. For two-proxy connections, we increase the window size to 500MB for intra-cloud communications only.

B. System Evaluations

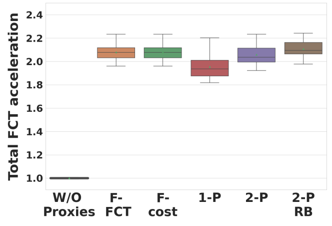
Impact of number of flows. We start by evaluating the impact of the number of flows on performance in a one-to-many use case. One source in Tokyo transfers data to each destination. At each step, we increment the number of flows by randomly adding a new destination worldwide. We set the budget proportionally to the number of flows. Fig. 6(a) shows that proxy-greedy algorithms improve the total FCT in the system and outperform the flow-greedy algorithms. Fig. 6(b) illustrates how proxy-greedy algorithms use less proxies. Then, Fig. 6(c) shows that this is because proxy-greedy algorithms prefer having many flows share a single proxy for one-proxy acceleration. By saving on the proxy setup cost, they can accelerate more additional flows. By contrast, flow-greedy algorithms rely on expensive two-proxy acceleration.

Impact of budget. Fig. 7(a) shows the influence of budget on the overall FCT improvement for each algorithm. We keep the settings of the previous one-to-many evaluation and consider 30 flows. We can see that the proxy-greedy algorithms are superior for low and medium budgets. With a high budget, the F-FCT algorithm gets the best result, because it always picks the best proxy locations regardless of cost.

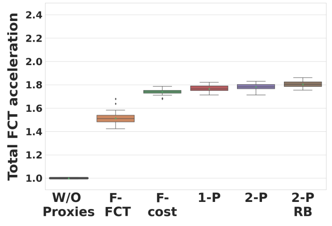
Impact of cost parameters. Fig. 7(b) and 7(b) show the impact of cost parameters. Each boxplot box represents the results between the 25th and 75th percentiles of 30 runs. Fig. 7(b) doubles the data-transfer cost to 16¢ per GB and zeroes the proxy-setup cost. All algorithms get similar results based on two-proxy acceleration since there is no cost for setting proxies, except for the 1-P algorithm which is less able to place efficiently the corresponding two proxies. Fig. 7(c) zeroes the data-transfer cost and doubles the proxy-setup cost



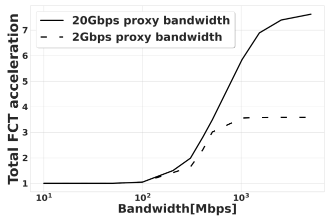
(a) Budget impact



(b) Data-transfer cost impact

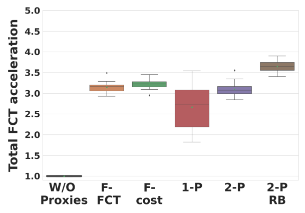


(c) Proxy set-up cost impact

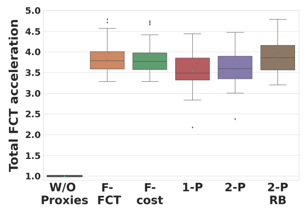


(d) Access-bandwidth impact

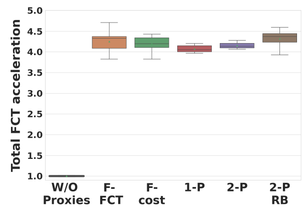
Fig. 7: Impact of parameters on overall FCT acceleration. (a) Impact of budget: The proxy-greedy family better leverages low budgets and achieves higher accelerations. On the other hand, with high budgets, the F-greedy FCT that tends to pick the best and most expensive two-proxy acceleration choices manages to achieve the unlimited-budget bound, while the other algorithms cannot improve their greedily-picked choices. (b) Impact of cost: With dominant data-transfer costs, both families get similar results. When there is no cost for setting proxies, all algorithms use two-proxy acceleration. (c) With dominant proxy-setup cost, the cost-efficient algorithms obtain better accelerations. (d) Impact of last-mile access bandwidth: As it grows, FCT acceleration increases significantly, especially when the cloud-proxy capacity is 20Gbps.



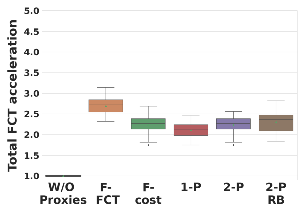
(a) One-to-many



(b) Many-to-many (full mesh)



(c) Many one-to-one



(d) Many localized one-to-many (CDN)

Fig. 8: Impact of use cases with 50¢ per-flow budget. As expected, we get a strong acceleration for the first three use cases. In the fourth case that exploits CDN localization, smaller distances enable less proxy options and therefore a lower acceleration.

to \$1. Proxy-sharing cost-efficient algorithms perform better.

Impact of last-mile access bandwidth. Fig. 7(d) shows the impact of the last-mile access bandwidth on the total acceleration. We use the 2-P RB algorithm, assume 50¢ per flow, and compare two types of proxy: small (2Gbps), as in current cheapest proxies, and large (20Gbps), assuming next-generation proxies will have larger limits. As the access bandwidth grows beyond some 100Mbps, the FCT acceleration increases significantly, especially with the large proxy capacity. This is because the flows are less constrained by the last-mile bandwidth, but rather by the long RTT, in which case cloud proxies with TCP splitting help more. This may partly justify the current increased interest in cloud proxies, as last-mile fiber-optics deployment becomes wider.

Use cases. Fig. 8 shows the algorithm performance results for all four different use cases of §III. In all cases, we assume a 50¢ budget per flow and measure the total-FCT improvement for 60 flows. In the *many localized one-to-many* CDN-like topology, we first randomly select three sources in three different continents: Asia, Europe and North America. Since CDNs are not perfect, at each step, when we sample a random destination, it connects to its closest source with 90% probability, to its second-closest source with 7% probability, and to its farthest source with 3%. The first three use cases get high accelerations. In the fourth, the acceleration is smaller due to the shorter average distance, but non-negligible due to the many available proxy locations that enable us to perform a two-proxy acceleration (especially with F-FCT).

Run-time. The simulation time for an example run with 60 flows in unoptimized Python, as in Fig. 6(c), takes less than one second for the flow-greedy algorithms while for the 1-P, 2P, and 2P with RB, it takes $1.6s$, $6.5s$, and $131s$, respectively.

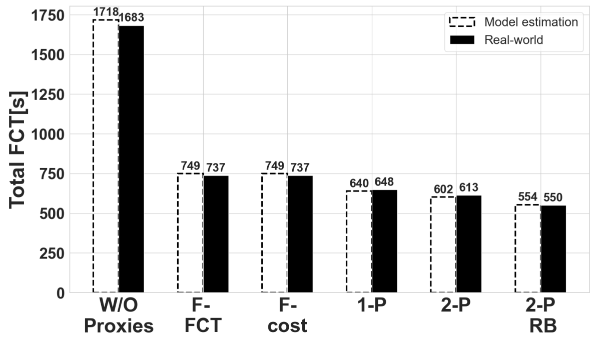


Fig. 9: Real-world experiment: We measure the total FCT obtained in a real-world experiment on GCP. We consider a many-to-many topology with users in Brazil, England, Finland, and Japan. We use our CloudPilot system to deploy cloud proxies with Kubernetes and HAProxy, such that our algorithms select the locations. We then compare the results against our model prediction. We can see that our FCT acceleration prediction achieves very close results to the obtained real-world results.

C. Cloud Experiments

Methodology. We deploy our CloudPilot system on GCP to set up the proxies and run iPerf3 tests (details in §IV). We consider a many-to-many full-mesh use case with four servers in Hamina (Finland), London (England), Sao-Paulo (Brazil), and Tokyo (Japan), and therefore twelve flows, and measure the FCT of each flow. All the hosts run virtual machines with default instances (E2-medium). The budget is 6\$, *i.e.,* 50¢ per flow. Each result averages 20 runs.

Results. Fig. 9 shows how the proxy-greedy algorithms achieve better real-world results than the flow-greedy ones, as previously seen in the simulations. Significantly, as we also saw in the model evaluation for individual flows (Fig. 3), our modeled predictions for the total system FCT appear close to the real-world measured FCTs.

VIII. Conclusion

In this paper, we introduced *CloudPilot*, a Kubernetes-based system that measures communication parameters across different cloud regions, and uses these measurements to deploy cloud proxies in optimized locations on multiple cloud providers. We further demonstrated how it can significantly improve the flow completion time of global geo-distributed applications by relying on an optimized placement of cloud proxies. In future work, we intend to expand on a speculative CloudPilot version that uses various prediction models to estimate the upcoming future patterns and their proxy needs.

Acknowledgment

The authors would like to thank Ofer Biran, Roy Mitrany and Aran Bergman for useful discussions. This work was partly supported by the Louis and Miriam Benjamin Chair in Computer-Communication Networks, the Israel Science Foundation (grant No. 1119/19), and the Hasso Plattner Institute Research School.

References

[1] B. Yeganeh *et al.*, "A first comparative characterization of multi-cloud connectivity in today's internet," in *Passive and Active Meas.*, 2020.

[2] T. K. Dang *et al.*, "Cloudy with a chance of short RTTs: analyzing cloud connectivity in the internet," in *ACM IMC*, 2021, pp. 62–79.

[3] R. K. Mok *et al.*, "Measuring the network performance of Google Cloud Platform," in *ACM IMC*, 2021, pp. 54–61.

[4] F. Lai, M. Chowdhury, and H. Madhyastha, "To relay or not to relay for Inter-Cloud transfers?" in *USENIX HotCloud*, Boston, MA, 2018.

[5] D. Bhattacherjee, M. Tirmazi, and A. Singla, "A cloud-based content gathering network," in *USENIX HotCloud*, Santa Clara, CA, 2017.

[6] O. Haq, C. Doucette, J. W. Byers, and F. R. Dogar, "Judicious QoS using cloud overlays," in *ACM CoNEXT*, 2020, pp. 371–385.

[7] A. Pathak *et al.*, "Measuring and evaluating TCP splitting for cloud services," in *PAM'10, Zurich, Switzerland*, April 2010.

[8] F. Le, E. Nahum, and D. Kandlur, "Understanding the performance and bottlenecks of cloud-routed overlay networks: A case study," in *ACM Workshop on Cloud-Assisted Networking*, 2016, p. 7–12.

[9] C. X. Cai, F. Le, X. Sun, G. G. Xie, H. Jamjoom, and R. H. Campbell, "CRONets: Cloud-routed overlay networks," in *IEEE ICDCS*, 2016.

[10] A. Bergman *et al.*, "Pied piper: Rethinking internet data delivery," *arXiv preprint arXiv:1812.05582*, 2018.

[11] A. Markuze, A. Bergman, C. Dar, I. Keslassy, and I. Cidon, "Kernels of splitting TCP in the clouds," in *Netdev 0x14*, 2020.

[12] R. Taft *et al.*, "Cockroachdb: The resilient geo-distributed SQL database," in *ACM SIGMOD*, 2020.

[13] F. Zhang *et al.*, "CBase: A new paradigm for fast virtual machine migration across data centers," in *IEEE/ACM CCGRID*, 2017.

[14] G. Pierre and M. van Steen, "Globule: a collaborative content delivery network," *IEEE Communications Magazine*, 2006.

[15] (2022) CloudPilot system git. [Online]. Available: https://github.com/kfirtoledo/CloudPilot_Project

[16] M. Luglio, M. Sanadidi, M. Gerla, and J. Stepanek, "On-board satellite "split TCP" proxy," *IEEE J. Select. Areas Commun.*, 2004.

[17] V. Farkas, B. Héder, and S. Nováczki, "A split connection TCP Proxy in LTE Networks," in *Information and Comm. Technologies*, 2012.

[18] B. H. Kim, D. Calin, and I. Lee, "Enhanced split TCP with end-to-end protocol semantics over wireless networks," in *IEEE WCNC*, 2017.

[19] N. H. Rotman *et al.*, "Cloudcast: Characterizing public clouds connectivity," *arXiv preprint arXiv:2201.06989*, 2022.

[20] H. Zhang *et al.*, "Harmony: An approach for geo-distributed processing of big-data applications," in *IEEE CLUSTER*, 2019.

[21] Q. Pu *et al.*, "Low latency geo-distributed data analytics," *ACM SIGCOMM CCR*, 2015.

[22] K. Kloudas *et al.*, "Pixida: Optimizing data parallel jobs in wide-area data analytics," *Proc. VLDB Endow.*, 2015.

[23] P. Li *et al.*, "Traffic-aware geo-distributed big data analytics with predictable job completion time," *IEEE TPDS*, 2017.

[24] A. Jonathan *et al.*, "Nebula: Distributed edge cloud for data intensive computing," *IEEE TPDS*, 2017.

[25] Y. Guo, Z. Ge, B. Urgaonkar, P. Shenoy, and D. Towsley, "Dynamic cache reconfiguration strategies for a cluster-based streaming proxy," in *Web Content Caching and Distribution*, 2004.

[26] J. Wu and K. Ravindran, "Optimization algorithms for proxy server placement in content distribution networks," in *IFIP/IEEE International Symposium on Integrated Network Management-Workshops*, 2009.

[27] C. Ge *et al.*, "QoE-assured 4K HTTP live streaming via transient segment holding at mobile edge," *IEEE J. Select. Areas Commun.*, 2018.

[28] J. Deng *et al.*, "Internet scale user-generated live video streaming: The Twitch case," in *Passive and Active Measurement*, 2017, pp. 60–71.

[29] (2022) iPerf3 - measuring network performance tool. Accessed: 2022-30-01. [Online]. Available: https://iperf.fr/iperf-download.php

[30] F. Kelly, "Mathematical modelling of the internet," in *Mathematics unlimited—2001 and beyond*. Springer, 2001, pp. 685–702.

[31] (2022) Public SpeedTest servers. [Online]. Available: https://as62240.net/speedtest

[32] (2022) GCP inter region latency. [Online]. Available: https://docs.aviatrix.com/HowTos/gcp_inter_region_latency.html

[33] K. Lepikhov. (2022) Propagation delay in Géant. [Online]. Available: https://wiki.geant.org/display/public/EK/PropagationDelay

[34] (2022) GCP network pricing. [Online]. Available: https://cloud.google.com/vpc/pricing

[35] (2022) GCP VM pricing. [Online]. Available: https://cloud.google.com/pricing/list

[36] (2022) GCP-machine family description. [Online]. Available: https://cloud.google.com/compute/docs/general-purpose-machines

[37] C. F. Lam, "(invited) Google Fiber Deployments: Lessons learned and future directions," in *OFC*, 2021.

[38] S. Martello and P. Toth, "Solution of the zero-one multiple knapsack problem," *European J. of Op. Research*, 1980.

IX. Appendix: Proof of Theorem 1

Proof: The 0/1 multiple-knapsack problem (MKP) [38] is a known NP-hard problem. In this problem, we need to place a subset of N non-splittable items in M bins. Each item i has a given positive weight w_i and profit p_i. The sum of the weights of all items in a bin j cannot exceed its capacity C_j. Our goal is to place a subset of items in the bins with maximum sum of the subset item profits.

Given any 0/1 MKP instance, we define a corresponding instance of the cloud-proxy placement problem with a single proxy location, and show that solving it would also solve the 0/1 MKP problem. We define N flows, and can freely choose their flow rates as w_i (we can arbitrarily change the maximum window, given an infinite BW and a fixed RTT), and flow FCT gain (difference between FCT in direct connection and FCT using the proxy) as p_i (we can arbitrarily change the data size of flow i). We define the budget as B. We set the bandwidth cost as $C_{BW} = 0$ and proxy-setup cost as $C_{setup} = \frac{B}{M}$, so the budget allows exactly M proxy instances at this location. We set the bandwidth capacity of proxy j to C_j. Since there is only one proxy location, using two-proxy acceleration is never beneficial. If there is a solution to our problem, we can also solve the 0/1 MKP. Hence, by reducing the 0/1 MKP to the above problem, we find it is NP-Hard. ■