








# Towards Optimizing the Edge-to-Cloud Continuum Resource Allocation

Igor Ferrazza Capeletti<sup>3</sup><sup>a</sup>, Ariel Goes de Castro<sup>3</sup><sup>b</sup>, Daniel Chaves Temp<sup>1,3</sup><sup>c</sup>,  
Paulo Silas Severo de Souza<sup>2</sup><sup>d</sup>, Arthur Francisco Lorenzon<sup>4</sup><sup>e</sup>, Fábio Diniz Rossi<sup>1,3</sup><sup>f</sup>  
and Marcelo Caggiani Luizelli<sup>3</sup><sup>g</sup>

<sup>1</sup>*Federal Institute Farroupilha, Alegrete, Brazil*

<sup>2</sup>*Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil*

<sup>3</sup>*Federal University of Pampa, Alegrete, Brazil*

<sup>4</sup>*Federal University of Rio Grande do Sul, Porto Alegre, Brazil*

**Keywords:** Cloud Continuum, Resource Allocation, Heuristics, Simulation.

**Abstract:** The IT community has witnessed a transition towards the cooperation of two major paradigms, Cloud Computing and Edge Computing, paving the way to a Cloud Continuum, where computation can be performed at the various network levels. While this model widens the provisioning possibilities, choosing the most cost-efficient processing location is not trivial. In addition, network bottlenecks between end users and computing facilities assigned for carrying out processing can undermine application performance. To overcome this challenge, this paper presents a novel algorithm that leverages a path-aware heuristic approach to opportunistically process application requests on compute devices along the network path. Once intermediate hosts process information, requests are sent back to users, alleviating the demand on the network core and minimizing end-to-end application latency. Simulated experiments demonstrate that our approach outperforms baseline routing strategies by a factor of 24x in terms of network saturation reduction without sacrificing application latency.


## 1 INTRODUCTION


Cloud Continuum is an emerging paradigm that extends the traditional Cloud to the Edge, Fog, and in-between. In other words, it is an aggregation of heterogeneous resources of other computing facilities, such as micro-data centers and intermediate computing nodes along the path between the user's requests and larger computing premises (Moreschini et al., 2022b; Liao et al., 2022).


By extending the computing capabilities, Cloud Continuum allows the in-transit processing of application requests and expands the computing capabilities (and granularity) (Baresi et al., 2019). There are many advantages of this computing paradigm. For instance, application requests can be processed while


routed to the following processing hop (e.g., an Edge node). By doing that, it can reduce communication latency and improve the resource utilization of Cloud and Edge nodes by freeing their resources up. It is imperative for emerging applications with stringent performance requirements in a highly connected and dynamic environment. Examples of such applications include multi-sensory extended reality, brain-computer interfaces, haptic interaction, meta-verses, and flying vehicles (Yeh et al., 2022).


Figure 1 illustrates a Cloud Continuum example. In the traditional Cloud Computing approach, application requests (e.g.,  $R_3$ ) are processed by a centralized Cloud server. In the Cloud Continuum, however, computing resources are spread over the network infrastructure. That includes, for instance, Edge Cloud nodes and micro-servers placed closer to the access network (e.g., base stations). However, the resources available at these computing premises are usually quite limited. In the example, request  $R_2$  could be processed entirely by an Edge Cloud server (in case it has enough resources) or, depending on the request's requirements, still be processed partially by different Edge Cloud servers. In turn, request  $R_2$  is to-


<sup>a</sup>  <https://orcid.org/0000-0002-8712-2195>


<sup>b</sup>  <https://orcid.org/0000-0002-5391-5082>

<sup>c</sup>  <https://orcid.org/0000-0002-9724-1331>

<sup>d</sup>  <https://orcid.org/0000-0003-4945-3329>

<sup>e</sup>  <https://orcid.org/0000-0002-2412-3027>

<sup>f</sup>  <https://orcid.org/0000-0002-2450-1024>

<sup>g</sup>  <https://orcid.org/0000-0003-0537-3052>

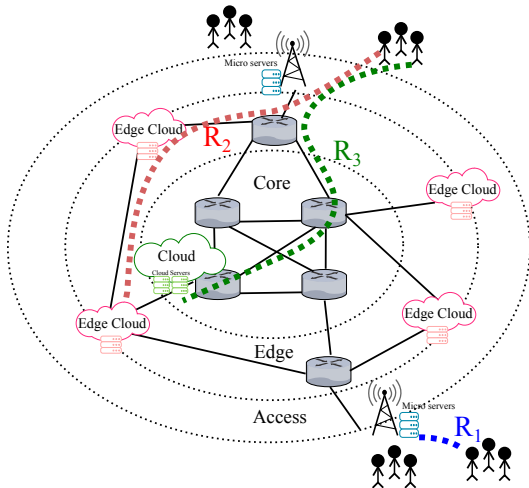


Figure 1: Overview of the Cloud-Continuum Computing.

tally processed by a micro-server located at the base station at one-hop from the users.

Despite the extra layer of available resources provided by the aggregation of heterogeneous resources in the infrastructure, it is challenging to efficiently use distributed computing resources without proper orchestration. For instance, a Cloud Platform can route requests between Edge and Cloud nodes by using consolidated routing protocols (e.g., OSPF (Fortz and Thorup, 2000) – based on the shortest path). Even applying customized routing schemes to improve other network metrics (e.g., to balance network link load) might under-use the available computing resources in the network. An efficient solution would be to route application requests using a path with higher access to computation power. In this case, a request could be processed opportunistically by a computing premise along the path. In the best case, the request is processed in the first hop. Otherwise, it is routed to its final destination (e.g., the Cloud), passing through nodes that have direct access to more powerful computing premises (e.g., Edge Cloud servers).

This paper proposes a path-aware heuristic approach to efficiently route application requests to computing nodes to fill this gap. The main advantage of our heuristic approach is that it considers the available resources along the path of the network to process the requests as early as possible. If a computing premise along the path processes a request, it is routed back to the source device. Our heuristic approach is based on the  $k$ -shortest path. However, alternative paths obtained by our search algorithm try not to detour much from the original shortest path. It does that by incrementally allowing an expansion of the neighborhood of nodes in the original path. Results show that our solution outperforms shortest-path-based so-

lutions by a factor of 24X in terms of the number of processed applications while not imposing significant delay. Our contributions can be summarized as follows:

- We formalize an orchestration model for in-transit processing on Cloud Continuum environments.
- We propose a novel algorithm that opportunistically processes application requests on compute devices along the Cloud Continuum.
- We present an evaluation showing that our algorithm reduces the network saturation by 24x compared to traditional routing strategies.
- We disclose the dataset and source code of our approach to foster reproducibility.

The remainder of this paper is organized as follows. In Section 2, we provide a brief background on Cloud Continuum aspects. Then, we discuss the related literature in the area. In Sections 3 and 4, we introduce our model and present the proposed heuristic. In Section 5, we discuss the obtained results. Last, in Section 6, we conclude the paper with final remarks and perspectives for future work.

## 2 BACKGROUND AND RELATED WORK

In this section, we start by discussing cloud computing concepts. Next, we overview the most prominent work related to Cloud Continuum computing.

### 2.1 Cloud Computing and Beyond

Cloud computing is a paradigm that allows users to move out their data/applications from local computing to remote “cloud” (Wang et al., 2010). There are many benefits to this approach. First, estimating the cost of acquiring new equipment (e.g., switches and servers) is not trivial. Consider certain services may have different demands throughout the day – i.e., peak hours. In this case, a budget can end up being (i) overestimated, wasting resources, or (ii) underestimated and needing to be more to deal with specific user demands, such as enough bandwidth or processing capacity to deal with an excessive amount of requests. Second, the equipment takes up space and is costly. It is necessary to dedicate considerable time to guarantee the correct functioning of the equipment and its running services to maintain qualified staff in the payroll for unforeseen problems. Finally, planning the above goals is time-consuming, and managers could use this time for other company tasks.

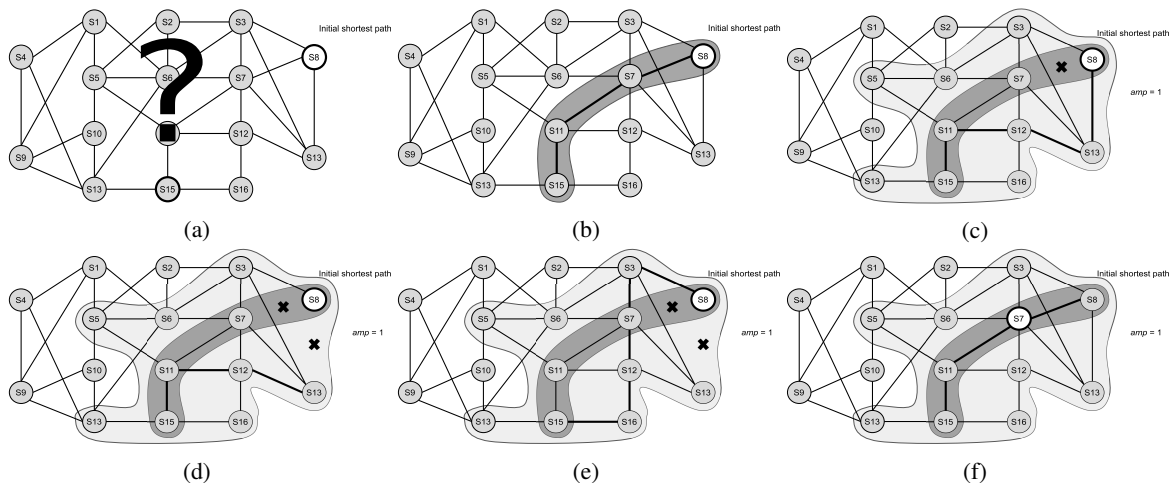


Figure 2: Illustration of the behavior of our heuristic approach when finding near-optimal solutions.

Despite that, the cloud computing model struggles to keep up with the constraints of current services, such as ultra-low latency services in 5G cellular networks (Chiu et al., 2016). To tackle this problem, new computation paradigms such as edge computing (Satyanarayanan, 2019) arise to satisfy these demands. More specifically, edge computing allows performing computation and storage closer to the end-user with minimal to no intervention from cloud nodes - incurring less latency. Despite that, edge computing still has the limited processing power and high-time responses. It may require cooperation from remote resources such as those in cloud provider infrastructure that do not cope well with these requirements.

More recently, the cloud continuum has drawn attention from academia and industry as a candidate to overcome these limitations and has several definitions (Moreschini et al., 2022a). In this paper, we consider an early definition by Gupta et al (Gupta et al., 2016) where it defines cloud continuum as “a continuum of resources available from the network edge to the cloud/datacenter”. That said, with the advent of programmable network devices (e.g., SmartNICs, programmable switches), the offloading of applications to the data plane presents a superior performance in terms of high-throughput, low-latency computing which may be combined with the existing edge infrastructure to avoid the need to access high-latency computation resources located in centralized clouds - to guarantee SLAs are met.

## 2.2 Related Work

This section discusses the most prominent studies related to in-transit or in-path computing strategies.

NetCache (Jin et al., 2017) leverages switch

ASICs to perform on-path network caching to store key-value data. Similarly, Wang et al. (Wang et al., 2019) is the first to design and implement packet aggregation/disaggregation entirely in switching ASICs while PMNet (Seemakhupt et al., 2021) persistently store and update data in network devices with sub-RTT latency. SwitchTree (Lee and Singh, 2020) estimates flow-level stateful features, such as RTT and per-flow bitrate. FlexMon (Wang et al., 2022) presents a network-wide traffic measurement scheme that optimally deploys measurement nodes and uses these nodes to measure the flows collaboratively.

Tokusashi et al. (Tokusashi et al., 2019) selectively offload services to the data plane according to changes in workload. Similarly, Mai et al. (Mai et al., 2020) partially offloads the lightweight critical tasks to the data plane devices and leaves the rest to the mobile edge computing (MEC) nodes. In contrast, Saquetti et al. (Saquetti et al., 2021) distributes neuron computation of an Artificial Neural Network to multiple switches. Friday et al. (Friday et al., 2020) introduces an engine to detect attacks in real-time by analyzing one-way ingress traffic on the switch. Similarly, INDDos et al. (Ding et al., 2021) can reduce DDoS detection time. It identifies as attacks the destination IPs contacted by several source IPs greater than a threshold, in a given time interval, entirely in the data plane. SmartWatch (Panda et al., 2021) leverages advances in switch-based network telemetry platforms to process the bulk of the traffic and only forward suspicious traffic subsets to the SmartNIC, which have more processing power to provide finer-grained analysis.

Kannan et al (Kannan et al., 2019) is the first to perform time synchronization in the data plane, enabling to add of high-resolution timing information to the packets at line rates. Tang et al (Tang

et al., 2020) and pHeavy (Zhang et al., 2021) make efforts to reduce the time to detect heavy hitters in the data plane. (Tang et al., 2020) propose a compact sketch statically allocated in switch memory, while (Zhang et al., 2021) introduces a machine learning-based scheme to mitigate latency overhead to SDN controllers. (Sankaran et al., 2021) increases data plane security by restricting modifications to a persistent network switch state, enabling ML decision-making computation to be offloaded to industrial network elements. Similarly, Kottur et al (Kottur et al., 2022) propose crypto externs for Netronome Agilo smartNICs for authentication and confidentiality directly in the data plane.

Most recent initiatives have focused on offloading computing mechanisms to specific computing premises, such as programmable network devices. In this work, we take a further step toward efficiently using distributed resources in the Edge-to-Cloud Continuum to increase the network capability regarding the number of processed application requests.

### 3 SYSTEM MODEL

This section describes the resource allocation model proposed in this work for the Edge-to-Cloud Continuum approach. Next, we describe the inputs and outputs of our model, as well as the constraints and objective function. Table 1 summarizes the notations used hereafter.

#### 3.1 Model Description and Notation

**Input.** The optimization model considers as input a physical network infrastructure  $G = (D, L)$ , and a set of application requests  $A$ . Set  $D$  in network  $G$  represents routing/forwarding devices  $D = \{1, \dots, |D|\}$ , while set  $L$  consists of unidirectional links interconnecting pair of devices  $(i, j) \in (D \times D)$ . We assume that only one computing premise (e.g., Edge node) is connected to a forwarding device  $D$ . Devices  $D^c \subseteq D$  represents the subset of devices with computing premises. Each computing premise  $d \in D^c$  has a computational capacity defined by  $C : D^c \rightarrow \mathbb{N}^+$ . Conversely, each application  $i \in A$  has a computing requirement defined by  $R : A \rightarrow \mathbb{N}^+$ . We denote the routing taken by application  $i \in A$  as function  $\mathcal{P} : A \rightarrow \{D_1 \times \dots \times D_{|D-1}|\}$ . We assume the path given by function  $\mathcal{C}$  is simple.

For simplicity, we assume that distributed computing platforms can partially compute the computing power required by an application  $i \in A$ . As a simplification, we assume that partially computed values are

embedded into the packet that transports the request. Example of similar strategies that utilize the packet encapsulation to carry information includes In-Band Network Telemetry (INT) (Marques et al., 2019)(Hohemberger et al., 2019).

**Variables.** Our model considers a variable set  $X = \{x_{i,j}, \forall i \in A, j \in D\}$  which indicates the amount used by computing premises  $j \in D^c$  to process application  $i \in A$ .

$$x_{i,j} = \begin{cases} \mathbb{N}^+ & \text{If application } i \in A \text{ is processed by } j \in D^c \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

**Constraints.** Next, we describe the main feasibility constraints related to the optimization problem. The problem is subject to two main constraints: (i) path computing capacity and (ii) route connectivity constraints.

(i) *Path Computing Capacity:* Application  $i \in A$  has a computing requirement that is attended along the path taken by the application. Therefore, the routing path (or the computing path) establishes an upper bound of computing power. In other words, the amount of computing power in the path cannot be lower than the application requirement. Thus, in Equation set (2), we sum the available capacity along the routing taken by  $i$  and ensure it is equal to the application's requirement. Similarly, Equation set (3) ensures that the computing power of computing premises  $j \in D^c$  is not violated.

$$\sum_{\forall j \in \mathcal{P}(i): j \in D^c} C(j) \cdot x_{i,j} = R(i) \quad : (\forall i \in A) \quad (2)$$

$$\sum_{\forall i \in A, \mathcal{P}(i)} R(j) \cdot x_{i,j} \leq C(j) \quad : (\forall j \in D^c) \quad (3)$$

(ii) *Route Connectivity:* we assume all devices  $i \in \mathcal{P}(i)$  are pairwise strongly connected, i.e., any pair of devices in  $\mathcal{P}(i)$  are reachable to each other by a link  $(i, j) \in L$ . To describe this property, we recall a auxiliary function  $\delta : (\mathcal{P} \times D \times D) \rightarrow \{true, false\}$  that returns *true* in case there exists a path between node  $i$  and  $j$  in path  $\mathcal{C}$ . Note that one can use other constraints to describe route connectivity such as the based on flow conservation constraints. In other words,  $\mathcal{P}(i)_{[k]} \rightarrow \dots \rightarrow \mathcal{P}(i)_{[|\mathcal{P}|]}$ , where  $(\mathcal{P}(i)_{[k]}, \mathcal{P}(i)_{[k+1]}) \in L$ . Otherwise, function  $\delta$  returns *false*. Equation set (4) ensures all applications  $i \in A$  have a computing path, while Equation set (5) ensures they are valid (or connected).

$$|\mathcal{C}(i)| \neq \emptyset \quad : (\forall i \in A) \quad (4)$$

$$\delta(\mathcal{P}(i), k, l) = true \quad : (\forall i \in A), \forall (k, l) \in \mathcal{P}(i) \quad (5)$$



Table 1: Summary of symbols.

Symbol	Definiton
$G = (D, L)$	Physical infrastructure $G$ .
$D$	Set of forwarding devices.
$D^c$	Set of computing premises.
$L$	Set of physical links.
$A$	Set of applications.
$\mathcal{P}(i)$	Computing path given to application $i \in A$
$C : A \rightarrow \mathbb{N}^+$	Computing capacity of $j \in D^c$
$R : A \rightarrow \mathbb{N}^+$	Computing requirement of application $i \in A$

Given the feasibility constraints defined above, we assume there exists an assignment function  $\mathcal{A} : (G, i) \rightarrow (\mathcal{P}(i)) : \forall i \in A$  that, given a network infrastructure  $G$ , and a set of application  $A$ , it returns a feasible computing path ( $\mathcal{P}$ ), with respect to constraint sets (i) and (ii).

## 4 PROPOSED HEURISTIC

We propose a heuristic procedure that builds computation-aware paths to tackle the above problem efficiently and provide a quality-wise solution. Next, we overview the ideas behind our proposed heuristic and discuss the pseudo-code.

Consider Figure 2a illustrates a network topology  $\mathcal{G}(\mathcal{D}, \mathcal{L})$  where  $\mathcal{D}$  is a set of devices, and  $\mathcal{L}$  is the set of network links. Every time an end-user submits an application request, it must be routed somehow to reach its resources (e.g., the Cloud). If we wanted to orchestrate how to route an application from a host connected at the edge (e.g.,  $S8$ ) to a remote application (e.g.,  $S15$ ), there would be several strategies to reach this goal. Figures 2b to 2c summarize our proposal to solve this problem. First, we find the first shortest path from the origin of the request  $S8$  – which is the reference path –, leveraging existing programmable switches to reduce server computation overhead to the application server  $S15$ . In a binary manner, it is checked if the devices along the shortest path have enough resources (together) to completely process the network request entirely in the data plane (Figure 2b). Otherwise, the reference path ( $S8$ ,  $S7$ ,  $S11$ ,  $S15$ ) is iteratively modified according to the neighborhood size – controlled by the  $amp$  variable – as seen in Figure 2c. More specifically, we fix a node in the reference path and explore its adjacent neighbors. Then, detours are performed by applying the shortest path on sub-graphs that do not contain previ-

ously explored nodes (Figure 2c and Figure 2e). Finally, suppose none of the generated paths can fully offload the application request. In that case, we start the procedure all over again by fixing another node in the reference path, i.e., –  $S7$  as in Figure 2f.

Algorithm 1 summarizes the main procedure. For each application ( $app$ ), we store the shortest path (line 3) from a host connected to the source switch ( $s$ ) to the destination server ( $d$ ). If the switches in the path between  $s$  and  $d$  are unable to completely satisfy the application’s computing request ( $req$ ), we set the distance by the number of hops between the shortest path and the remaining graph (line 6) and narrows the search field for alternate switches based on  $amp$  value (line 8) - the higher, the broader. Then, an optimization procedure (line 10) is invoked to try and find a path to satisfy the computation request (see Algorithm 2 for details). Finally, if the optimization procedure succeeds, we store the new path. Otherwise, keep the default path.

The core of our proposal is presented in Algorithm 2. It iteratively tries to completely offload the server computation in the data plane by modifying the original path. For a given application  $app$ , we need its previous computation path  $cam\_old_{app}$ .

Algorithm 1: Overview of the procedure.

---

**Input:**  $\mathcal{G}(\mathcal{D}, \mathcal{L})$ : topology graph,  $\mathcal{A}$ : set of applications,  $amp$ : length of the subgraph.

- 1:  $graph_{old} \leftarrow graph$
- 2: **for**  $app$  in  $\mathcal{A}$  **do**
- 3:  $cam\_old_{app} \leftarrow dijkstra_{mod}(s, d, amp, graph)$
- 4: **if**  $sum_{cap}(cam_{app}) < req$  **then**
- 5:     **for**  $i \in cam_{app}$  **do**
- 6:          $set\_weights(amp, graph)$
- 7:     **end for**
- 8:      $graph \leftarrow gen\_subgraph(graph, amp)$
- 9:     **end if**
- 10:  $cam_{app} \leftarrow opt(app, graph, cam\_old_{app})$
- 11: **if**  $sum_{app}(cam_{app}) \neq NULL$  **then**
- 12:      $res \leftarrow add(cam_{app})$
- 13: **else**
- 14:      $res \leftarrow add(cam\_old_{app})$
- 15: **end if**
- 16:  $graph \leftarrow graph_{old}$
- 17: **end for**

---

The procedure works as follows: first, we (i) mark each node (line 1) as the reference  $ref$  at a time, then (ii) we perform detours by deleting the edge (line 2) starting from the current reference node to its up-following neighbor in the original path. Then, we repeat the process (lines 5-15) for the remaining adjacent nodes (line 4). Finally, if the taken detours reach the total application request, return the modified path (line 4).

---

Algorithm 2: Overview of the optimization procedure.

---

**Input:**  $\mathcal{G}'(\mathcal{D}, \mathcal{L})$ : topology subgraph,  $app$ : application,  $amp$ : length of the subgraph,  $cam_{old_{app}}$ : reference application computation path.

```

1: for  $ref \in cam_{old_{app}}$  do
2:    $del(link(ref, ref \rightarrow next))$ 
3:    $adj\_list_{ref} \leftarrow get\_adj\_nodes(ref)$ 
4:    $cam_{alt} \leftarrow dijkstra_{mod}(s, d, amp)$ 
5:   while  $adj\_list_{ref} \neq NULL$  and  $cam_{alt} \neq NULL$  do
6:     if  $sum_{cap}(cam_{alt}) \geq req$  then return  $cam_{alt}$ 
7:     else
8:       for  $j \in cam_{alt}$  do
9:         if  $j \in adj\_list_{ref}$  then
10:            $del(link(ref, j))$ 
11:            $cam_{alt} \leftarrow dijkstra_{mod}(s, d, amp)$ 
12:         end if
13:       end for
14:     end if
15:   end while
16: end for
return  $NULL$ 

```

---

## 5 EVALUATION AND DISCUSSION

This section describes the experiments carried out to evaluate the proposed algorithm. First, we detail our setup and methodology (§5.1). Then, we discuss the achieved results (§5.2).

### 5.1 Setup

To evaluate and assess the performance metrics of our proposed heuristic algorithm, we implement it using Python language. All experiments were conducted on an AMD ThreadRipper 3990X with 64 physical cores and 32 GB of RAM, using the Ubuntu GNU/Linux Server 22.04 x86-64 operating system. For our experiments, we generated physical network infrastructures with 100 routers. We consider that each routing device in our network has a computing premise attached to it. Physical networks are generated randomly with the link connectivity probability ranging from 10% to 50%. Each network link has latency values that vary between 10ms and 100ms. All computing premises have a processing capacity between 200 and 500, while 100 applications are requested to demand processing power ranging from 100 to 200. The source and destination node of each request is generated randomly. We varied in our approach the parameter  $amp$  between 1 and 2. This parameter controls the search depth, as already discussed.

We repeated each experiment 30 times to obtain the average and ensure a confidence level of at least 95%.

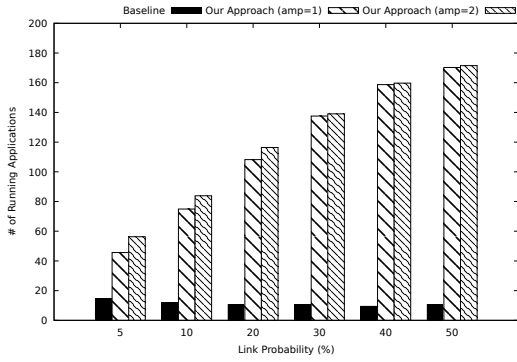
**Baseline.** We compare our approach against an OSPF-based approach. All requests follow the shortest path between source and destination nodes. We varied the order in which the application's requests are processed based on the computing power requested: random (*rnd*), ascending (*asc*), and descending (*dsc*).

### 5.2 Results

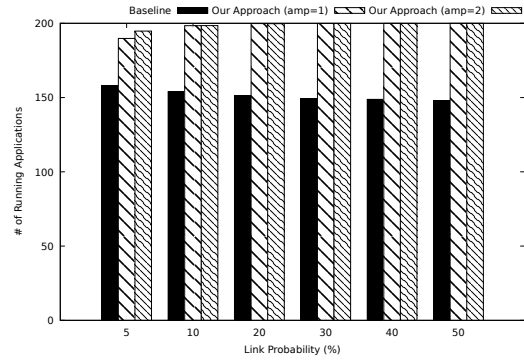
**Neighborhood Search.** Figure 3 illustrates how the value of  $amp$  impacts the shortest path computation on the search for distinct alternative paths. In the baseline ( $amp$  equal to 0), fewer applications are entirely offloaded to the computing premises for both categories (*1\_to\_1* and *n\_to\_1*) because there allowed no alternative routes besides the anchor path. On average, our approach offloaded 5% of the applications in the first category (Figure 3a). In contrast, it increases to 75% when compared to the second strategy (Figure 3b) – i.e., multi-source – because there are proportionally more paths for different applications. Also, for  $amp$  equal to 1 and 2 scenarios, our approach offloaded at least 1.2x more applications, up to 24.85x in the best case.

**Impact of Ordering Strategies.** Figure 4 illustrates the impact of applying different strategies to distribute the computation of a set of cloud applications into computing premises in the infrastructure. The *asc* algorithm orders application priority based on higher computation resources needed. On the other hand, the *dsc* strategy prioritizes less computation-intensive requests. Finally, the *rnd* strategy offloads requests as they arrive. We varied the source location (in the topology) where the requests were run. On the left (Figure 4a), the applications are limited to a single origin node, while on the right (Figure 4b), every node may be chosen randomly at each application request. We can observe that as the probability of the existence of a link increases for each pair of nodes (*x*-axis), the overall number of successfully processed applications also increases, reaching up to 173 running applications for the *rnd* strategy – i.e., applications are offloaded as they arrive. However, on the right, when we extend compute sources across the network (Figure 4b). In the worst case (5% link coverage topology), we already have 193 covered applications (i.e., 97% coverage) with the *dsc* strategy. It is because new shortest reference paths are created for each source-destination pair, and nodes from different reference paths can reach unreachable neighbors when the value of  $amp$  is too small for a single reference path.

**Path Length and Path Latency.** Figure 5a summa-

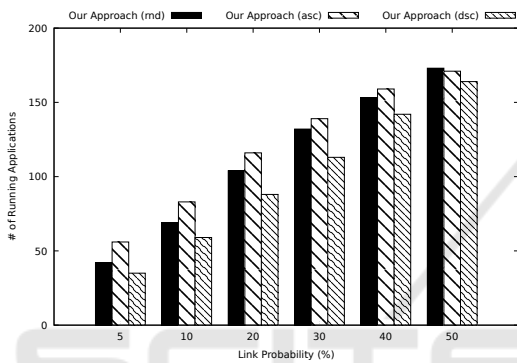


(a) Single source, single destiny.

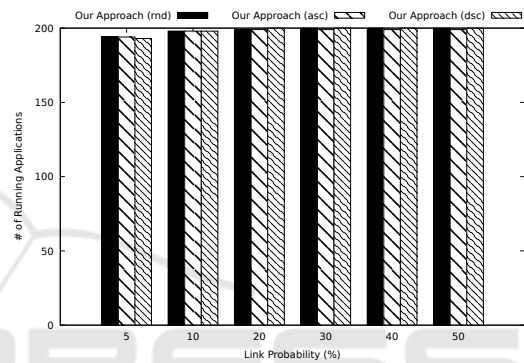


(b) Multiple sources, single destiny.

Figure 3: Impact of neighborhood search on the number of running applications.



(a) Single source, single destiny.



(b) Multiple sources, single destiny.

Figure 4: Impact of the order in which applications are processed on the number of running applications.

rizizes the average size of computation paths with a search radius on neighboring nodes fixed at up to 1 node - i.e.,  $amp$  being 1. - for single (Figure 5a) and multiple sources (Figure 5b). We can observe that the more link connections in both scenarios, the shorter the paths. As the nodes are more connected, it allows direct access to nodes with greater computing capacity, and thus it reduces the total path length. For example, when the link probability is doubled (from 5% to 10%), the path length decreases by 23.7% on average. It is even more evident when we have multiple paths because multiple anchor paths increase the probability that different neighbors also have a connection to a node with greater computing capacity. Also, the impact on the alternative path size is perceived in both scenarios. In the single source scenario, even with 50% link probability, the alternative path length is 25.7% longer than its anchor path. In parallel, this difference does not exceed 14.8% (with 50% link probability) for a multi-source scenario. On average, some applications have a chance of being resolved with fewer hops than using just one path. Similarly, Figure 5e indicates the cumulative latency for each path in milliseconds for both the single- and multi-source

runs. On average, single-source instances have more cumulative latency than multi-source runs. Finally, we can see a detailed look at the impact of latency in a single-source scenario on a CDF (Figure 5c). Similarly, a CDF shows resource availability decreasing as applications are allocated. In all strategies, network resources remain to be used after the allocation of applications. In this case, the random approach outperformed the others with 171 offloaded applications out of 200 in a single-source scenario (Figure 5f).

## 6 CONCLUSION AND FUTURE WORK

Cloud Computing has been in the spotlight for providing flexible and robust computing capabilities through the Internet (Buyya et al., 2009). The core component in the traditional Cloud model is the consolidation of computing resources on large-scale data centers, which comprise dedicated networks and specialized power supply and cooling mechanisms for maintaining the infrastructure.

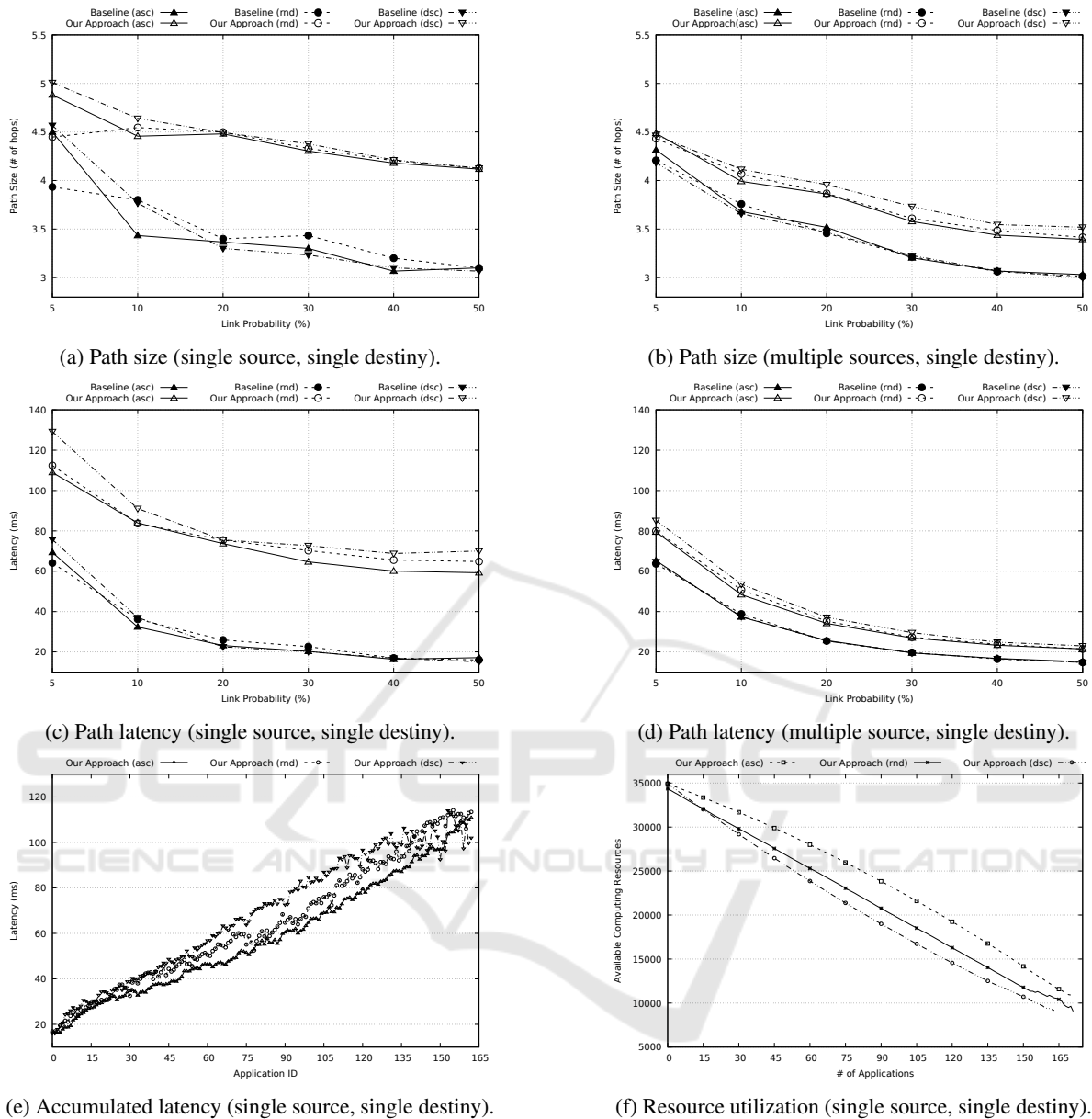


Figure 5: Average path latency and the number of hops per computing path.

As large-scale data centers require a complex and resource-consuming infrastructure, they typically cannot be deployed inside urban centers, where data sources are located (Satyanarayanan et al., 2019). On top of that, the emergence of applications with tight latency and bandwidth requirements has called into question Cloud’s prominence, highlighting the need for alternative approaches for processing the high data influx at reduced time. This challenge gave birth to the Cloud Continuum paradigm, which merges various paradigms, such as Cloud Computing and Edge Computing, to get the best-of-breed performance in

terms of latency and bandwidth.

There has been considerable prior work toward optimizing Cloud Continuum provisioning at its endpoints (i.e., on Cloud and Edge). However, we make a case for leveraging in-transit optimizations throughout the Cloud Continuum to mitigate performance issues. Despite a few initiatives in that line of reasoning, to the best of our knowledge, none of the existing approaches coordinates the in-transit application routing with the location of computing premises.

This paper presents a heuristic algorithm that orchestrates the application routing throughout the



Cloud Continuum, looking for suitable hosts for processing the requests along the way to reduce the applications' end-to-end delay. Simulated experiments demonstrate that the proposed solution outperforms baseline strategies by 24x in terms of served application requests without sacrificing the applications' delay. In future work, we intend to explore meta-heuristics and other optimization techniques to find optimal in-transit application schedules within a bounded time.

## ACKNOWLEDGEMENTS

This work was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) – Finance Code 001. Also, this work was partially funded by National Council for Scientific and Technological Development (CNPq 404027/2021-0), Foundation for Research of the State of Sao Paulo (FAPESP 2021/06981-0, 2021/00199-8, 2020/05183-0), and Foundation for Research of the State of Rio Grande do Sul (19/2551-0001266-7, 19/2551-0001224-1, 19/2551-0001689-1, 21/2551-0000688-9).

## REFERENCES

- Baresi, L., Mendonça, D. F., Garriga, M., Guinea, S., and Quattrocchi, G. (2019). A unified model for the mobile-edge-cloud continuum. *ACM Transactions on Internet Technology (TOIT)*, 19(2):1–21.
- Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616.
- Chiu, T.-C., Chung, W.-H., Pang, A.-C., Yu, Y.-J., and Yen, P.-H. (2016). Ultra-low latency service provision in 5g fog-radio access networks. In *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–6. IEEE.
- Ding, D., Savi, M., Pederzoli, F., Campanella, M., and Siracusa, D. (2021). In-network volumetric ddos victim identification using programmable commodity switches. *IEEE Transactions on Network and Service Management*, 18(2):1191–1202.
- Fortz, B. and Thorup, M. (2000). Internet traffic engineering by optimizing ospf weights. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, volume 2, pages 519–528 vol.2.
- Friday, K., Kfoury, E., Bou-Harb, E., and Crichigno, J. (2020). Towards a unified in-network ddos detection and mitigation strategy. In *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, pages 218–226. IEEE.
- Gupta, H., Nath, S. B., Chakraborty, S., and Ghosh, S. K. (2016). Sdfog: A software defined computing architecture for qos aware service orchestration over edge devices. *arXiv preprint arXiv:1609.01190*.
- Hohemberger, R., Castro, A. G., Vogt, F. G., Mansilha, R. B., Lorenzon, A. F., Rossi, F. D., and Luizelli, M. C. (2019). Orchestrating in-band data plane telemetry with machine learning. *IEEE Communications Letters*, 23(12):2247–2251.
- Jin, X., Li, X., Zhang, H., Soulé, R., Lee, J., Foster, N., Kim, C., and Stoica, I. (2017). Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 121–136.
- Kannan, P. G., Joshi, R., and Chan, M. C. (2019). Precise time-synchronization in the data-plane using programmable switching asics. In *Proceedings of the 2019 ACM Symposium on SDN Research*, pages 8–20.
- Kottur, S. Z., Kadiyala, K., Tammana, P., and Shah, R. (2022). Implementing chacha based crypto primitives on programmable smartnics. In *Proceedings of the ACM SIGCOMM Workshop on Formal Foundations and Security of Programmable Network Infrastructures*, pages 15–23.
- Lee, J.-H. and Singh, K. (2020). Switchtree: in-network computing and traffic analyses with random forests. *Neural Computing and Applications*, pages 1–12.
- Liao, Q., Marchenko, N., Hu, T., Kulics, P., and Ewe, L. (2022). Haru: Haptic augmented reality-assisted user-centric industrial network planning. In *2022 IEEE Globecom Workshops (GC Wkshps)*, pages 389–394. IEEE.
- Mai, T., Yao, H., Guo, S., and Liu, Y. (2020). In-network computing powered mobile edge: Toward high performance industrial iot. *IEEE network*, 35(1):289–295.
- Marques, J. A., Luizelli, M. C., Da Costa, R. I. T., and Gaspar, L. P. (2019). An optimization-based approach for efficient network monitoring using in-band network telemetry. *Journal of Internet Services and Applications*, (1):16.
- Moreschini, S., Pecorelli, F., Li, X., Naz, S., Hästbacka, D., and Taibi, D. (2022a). Cloud continuum: the definition. *IEEE Access*.
- Moreschini, S., Pecorelli, F., Li, X., Naz, S., Hästbacka, D., and Taibi, D. (2022b). Cloud continuum: The definition. *IEEE Access*, 10:131876–131886.
- Panda, S., Feng, Y., Kulkarni, S. G., Ramakrishnan, K., Duffield, N., and Bhuyan, L. N. (2021). Smartwatch: accurate traffic analysis and flow-state tracking for intrusion prevention using smartnics. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, pages 60–75.
- Sankaran, G. C., Sivalingam, K. M., and Gondaliya, H. (2021). P4 and netfpga based secure in-network computing architecture for ai-enabled industrial internet of things. *IEEE Internet of Things Journal*.

- Saqueti, M., Canofre, R., Lorenzon, A. F., Rossi, F. D., Azambuja, J. R., Cordeiro, W., and Luizelli, M. C. (2021). Toward in-network intelligence: running distributed artificial neural networks in the data plane. *IEEE Communications Letters*, 25(11):3551–3555.
- Satyanarayanan, M. (2019). How we created edge computing. *Nature Electronics*, 2(1):42–42.
- Satyanarayanan, M., Klas, G., Silva, M., and Mangiante, S. (2019). The seminal role of edge-native applications. In *International Conference on Edge Computing*, pages 33–40. IEEE.
- Seemakhupt, K., Liu, S., Senevirathne, Y., Shahbaz, M., and Khan, S. (2021). Pmnet: in-network data persistence. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 804–817. IEEE.
- Tang, L., Huang, Q., and Lee, P. P. (2020). A fast and compact invertible sketch for network-wide heavy flow detection. *IEEE/ACM Transactions on Networking*, 28(5):2350–2363.
- Tokusashi, Y., Dang, H. T., Pedone, F., Soulé, R., and Zilberman, N. (2019). The case for in-network computing on demand. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–16.
- Wang, L., Von Laszewski, G., Younge, A., He, X., Kunze, M., Tao, J., and Fu, C. (2010). Cloud computing: a perspective study. *New generation computing*, 28(2):137–146.
- Wang, S.-Y., Wu, C.-M., Lin, Y.-B., and Huang, C.-C. (2019). High-speed data-plane packet aggregation and disaggregation by p4 switches. *Journal of Network and Computer Applications*, 142:98–110.
- Wang, Y., Wang, X., Xu, S., He, C., Zhang, Y., Ren, J., and Yu, S. (2022). Flexmon: A flexible and fine-grained traffic monitor for programmable networks. *Journal of Network and Computer Applications*, 201:103344.
- Yeh, C., Do Jo, G., Ko, Y.-J., and Chung, H. K. (2022). Perspectives on 6g wireless communications. *ICT Express*.
- Zhang, X., Cui, L., Tso, F. P., and Jia, W. (2021). pheavy: Predicting heavy flows in the programmable data plane. *IEEE Transactions on Network and Service Management*, 18(4):4353–4364.