Latency-Aware Scheduling for Real-Time Application Support in Edge Computing

Kevin Röbert
Universität Hamburg
Hamburg, Germany
kevin.roebert@uni-hamburg.de

Heiko Bornholdt
Universität Hamburg
Hamburg, Germany
heiko.bornholdt@uni-hamburg.de

Mathias Fischer
Universität Hamburg
Hamburg, Germany
mathias.fischer@uni-hamburg.de

Janick Edinger
Universität Hamburg
Hamburg, Germany
janick.edinger@uni-hamburg.de

ABSTRACT

The relocation of computation from the network core to the edge where data is primarily generated has gained momentum, leading to the emergence of edge computing as a viable solution for low-latency processing. As a result, edge computing has the potential to significantly reduce response times, decrease bandwidth usage, enhance energy efficiency, and offer various other benefits. At the same time, end-user devices do not offer a consistent computing platform and Internet middleboxes severely restrict communication with edge devices. Often, this is circumvented by publicly accessible relay servers, which cause additional latency and render time-critical tasks unviable for offloading. This paper presents an approach capable of addressing the complexities inherent in edge computing and facilitating good decisions regarding latency-aware computation offloading. We conducted several real-world experiments to evaluate our approach and provide valuable data for further research. Our findings show that edge offloading is competitive to cloud and grid offloading, as it effectively reduces latency. Empirical evidence from our research supports that edge computing can offer significant advantages for real-time applications.

CCS CONCEPTS

• **Networks** → **Cloud computing**; **Network measurement**.

KEYWORDS

Edge computing, Cloud computing, Network measurement, Latency-awareness

ACM Reference Format:
Kevin Röbert, Heiko Bornholdt, Mathias Fischer, and Janick Edinger. 2023. Latency-Aware Scheduling for Real-Time Application Support in Edge Computing. In *6th International Workshop on Edge Systems, Analytics and Networking (EdgeSys '23), May 8, 2023, Rome, Italy.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3578354.3592866

EdgeSys '23, May 8, 2023, Rome, Italy

ACM ISBN 979-8-4007-0082-8/23/05…$15.00
https://doi.org/10.1145/3578354.3592866

1 INTRODUCTION

The increasing adoption of real-time applications, e.g., virtual and augmented reality, artificial intelligence, speech recognition, or autonomous vehicles, increases the demand for computing resources in potentially resource-constrained environments. Hence, these are perfect use cases for edge computing, but also impose strict requirements to computation offloading. With the advent of 5G technology, edge devices are anticipated to become increasingly compelling targets for offloading, owing to their potential to achieve latencies as low as 1 ms. The method presented in this paper offers an optimal means of exploiting such capabilities.

Based on subject tests [1], latencies of less than 150 ms are necessary for real-time audio applications. To reach a point where individuals cannot differentiate between actual speech and audio material, latencies less than 50 ms must be attained. Visual applications have even stricter latency requirements. The human eye can discriminate between 10–12 frames per second (FPS), perceiving higher frame rates as motion. However, movies and animations generally utilize at least 24 frames per second, which requires latencies of at most 41 ms. Besides the standard requirements such as speed and reliability, low latency and high throughput are becoming increasingly important for this type of application, resulting in significant challenges for traditional centralized cloud computing models. To overcome these challenges, edge computing has emerged as a promising solution by providing computing resources at the network edge to process data closer to the source. Edge offloading leverages computational resources and their proximity within organizations, public entities, or individual residences to provide real-time services while minimizing the latency. Despite the benefits of edge offloading, several challenges still need to be addressed, including edge devices' heterogeneity, availability, and reliability. In contrast to cloud and grid resources, end-user devices are not standardized, not always available nor dependable. Furthermore, middleboxes, e.g., firewalls and gateways doing network address translations, severely restrict communication with end devices. Consequently, a significant proportion of peers, up to 87% in contemporary P2P networks [2], require additional measures, e.g., network address translator (NAT) traversal via relay servers, to be reachable from the Internet. This measure usually increases the latency in between peers, as relay servers are deployed in cloud environments and thus negate the benefits of edge computing. Thus,

providing an abstraction layer capable of addressing the complexities of including end-user devices in edge computing, allowing to cope with restricted communication scenarios, and facilitating decisions regarding computation offloading is essential.

In this paper, we provide a comprehensive analysis of edge offloading and its associated latency behavior. For that, we conducted real-world experiments, whose results we describe in detail. In particular, we focus on end-user devices, which are most often located behind network barriers. Overcoming these barriers allows to include valuable resources for computation offloading in close vicinity and thus with low latency. We further empirically measured how simple priority-based scheduling strategies can enhance the service level of latency-sensitive applications in mixed (cloud/edge) environments. The findings indicate that edge offloading can significantly reduce latency in real-time applications compared to traditional centralized cloud computing models. The insights and findings presented in this paper can be used by researchers, practitioners, and system designers as a valuable resource for developing more realistic simulations that accurately reflect the performance of edge offloading in real-world scenarios.

The remainder of this paper is structured as follows. Related work is described in Section 2. Our system model for real-time edge computing is shown in Section 3. Following a real-world evaluation in Section 4 and a discussion of the results. Finally, Section 5 provides concluding remarks and future research directions.

2 RELATED WORK

Edge offloading has gained significant attention in research as an alternative to cloud offloading, which enables cloud computing capabilities at the edge of a network [3]. This approach allows for data locality, reduced bandwidth, and lower latencies. In their study, Corneo et al. [4] investigate the structure of in-network edge computing implementation and evaluate the potential reductions in latency that computational resources could offer to end-users in comparison to the pre-existing cloud infrastructure that has been deployed. In contrast to prior research, our study analyzes the effects on latency that arise from utilizing end-user devices at the edge of the network instead of relying on cloud resources. A formal study of edge service entity placement for VR applications was presented by Wang et al. in [5] together with combinatorial optimization showing its NP-hardness. For optimal, centralized solution for multi-user computation offloading in edge environments, Chen et al. [6] also showed its NP-hardness and presented a game theoretic approach. Scocal et al. [7] schedule tasks on edge, CDN, and cloud resources and evaluate their performance for latency-sensitive applications. However, the experiments were only simulated and not performed in a real environment. Mohan et al. [8] conducted client-to-cloud latency measurements by using RIPE Atlas. Nonetheless, the latency up to the local network of the edge devices was not taken into account. In [9], the performance of edge devices is analyzed in a real-world experiment. However, this work does not provide insight into latency behavior nor usability for real-time applications. Schäfer et al. [10] introduced the Tasklet middleware for computation offloading, which facilitates the distribution of computations to remote devices. In particular, the system envisions sharing idle resources from individuals or institutions. However, Tasklet lacks real-time capabilities.

Prior works frequently exhibit limited consideration of the actual conditions present at the edge with the users, thereby falling short in their representation of the same. Therefore, we provide a comprehensive analysis of edge offloading and its associated latency behavior through real-world experiments.

3 REAL-TIME EDGE COMPUTING

This section first presents the used middleware and then the strategies to make the edge real-time capable. For that, we describe methods to overcome various communication barriers such as firewalls, NATs, and internet gateway devices (IGDs), which make establishing connections with edge resources complex or time-intensive. Finally, we introduce priority-based scheduling strategies to improve latency-sensitive applications' service levels in mixed environments.

3.1 Middleware

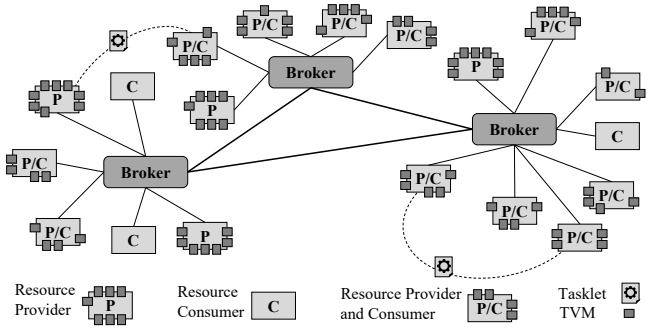


Figure 1: The Tasklet system architecture [10]: A hybrid approach to resource allocation, utilizing providers (P) to share idle capacities with consumers (C), broker-facilitated (B) resource management and matchmaking, and Tasklet-based workload distribution executed on virtual machines (TVM).

Our architectural design leverages the Tasklet middleware [10], that allows to distribute computations to remote devices. The system architecture is organized in a hybrid system, as depicted in Figure 1, comprised of three core components: *providers*, *consumers*, and *brokers*. The providers enable consumers to leverage their unused resources. Some devices might be provider and consumer at the same time. The brokers serve as resource managers, facilitating the matchmaking between providers and consumers. To effectively distribute computing tasks, consumers offload their workload in the form of Tasklets, which are self-contained units of computation at the granularity of function calls. These Tasklets typically have an execution time that ranges from hundreds of milliseconds to several minutes. The Tasklet middleware is executed on both the consumers' and providers' devices to provide transparent communication for application developers. The execution of Tasklets occurs on process-level virtual machines, known as *Tasklet Virtual Machines* (TVMs), which offer secure and isolated runtime environments while abstracting away from the underlying heterogeneous

hardware-software configurations of the devices. The Tasklet system is capable to deal with varying levels of reliability, availability, and performance of the resource providers. To ensure execution guarantees, the system introduces the quality of computation (QoC) concept, which is based on context-aware scheduling [11]. This QoC modifier becomes particularly important for applications that require a reliable or timely execution instead of a best-effort computing approach. Furthermore, the Tasklet system mechanisms enable the extension of serverless computing models, such as the Function-as-a-Service (FaaS) programming model, to edge systems. As a result, this framework represents an advancement in the development of edge computing architectures.

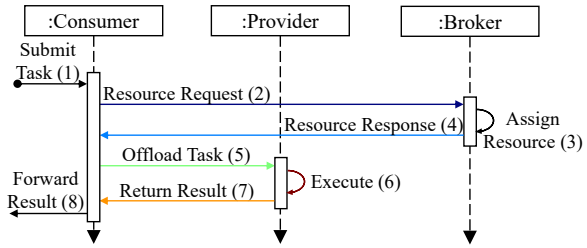


Figure 2: Tasklet life cycle.

The life cycle of a Tasklet is illustrated in Figure 2, and it comprises the following steps: (1) The application identifies a need for additional computing power and submits a Tasklet to the local running Tasklet middleware. (2) The middleware then sends a resource request to the broker. (3) The broker evaluates the available providers and selects the most appropriate one to fulfill the request. (4) The broker returns the address of the selected provider to the consumer. (5) The consumer then forwards the Tasklet directly to the selected provider. (6) The provider executes the Tasklet on one of its Tasklet Virtual Machines (TVMs). Once the Tasklet is executed, the provider (7) returns the execution result to the consumer. (8) The local middleware then relays the result to the application. This process enables efficient utilization of edge computing resources, and it helps to meet the resource demands of applications in a timely and effective manner.

3.2 NAT Traversal

To facilitate edge computation offloading, low-latency communication among providers, consumers, and brokers is essential. Ideally, the network structure should adhere to the Tasklet system architecture depicted in Figure 1, where providers and consumers can register with their preferred brokers and exchange Tasklets and results. However, the practical implementation of this ideal network structure is not always feasible on the Internet due to several factors: First, it requires each host to be equipped with a publicly routable Internet Protocol (IP) address and to be able to communicate bidirectionally with other hosts. However, the Internet comprises several independent, autonomous systems, often divided by communication barriers such as firewalls, NATs, and IGDs, making it difficult or impossible to establish connections with some hosts. As a result, certain hosts can only establish connections to others but cannot be reached via connections initiated by others. This unilateral restriction predominantly affects mobile, residential, or corporate networks, which renders the devices within these networks unreachable. Incorporating these barriers into the Tasklet system architecture implies that brokers must operate on public hosts that are reachable to all. In contrast, providers and consumers in edge environments may be able to register with these brokers but unable to exchange Tasklets or results.

The challenge of inaccessible edge devices is frequently addressed through relay servers. However, this solution introduces additional communication hops, resulting in higher latency. Additionally, deploying relay servers in cloud environments undermines the advantages offered by edge computing and can render computation offloading unviable. Therefore, we constructed an overlay that matches the ideal network structure (see Figure 3) required for low-latency offloading. To build this overlay, our approach uses the middlebox traversal technique presented by Bornholdt et al. [12] to render previously unreachable hosts reachable. In contrast to traditional middlebox traversal techniques such as Interactive Connectivity Establishment (ICE) [13], this traversal approach enables the secure establishment of direct connections with the lowest possible number of messages. This is achieved by performing handshake optimizations allowing to save 1–2 round-trip times (RTTs).

The mechanism of middlebox traversal involves initiating a connection by two devices to a known rendezvous server, which subsequently facilitates the exchange of network endpoint information between the devices. Using this information, the devices can establish direct communication via a temporary opening in their respective middlebox firewalls. This opening enables incoming traffic from the other device, eliminating the need for a third-party intermediary to relay communications between the two devices. However, despite the handshake optimizations, there are still 1–2 RTTs required to establish direct connections. Moreover, such a delay may exceed the acceptable limit in some real-time applications. Therefore, supplementary methods are needed to proactively establish connections to potential offloading targets. Thus, our approach proactively creates an overlay network structure allowing all providers, consumers, and brokers to reach each other. Once the overlay is established, the Tasklet system can offload as usual without additional communication overhead.

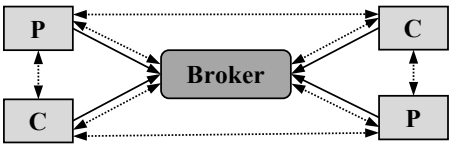


Figure 3: Without an overlay, only the broker can be contacted (solid arrows). Dashed arrows symbolize using an overlay-enabled communication protocol that aligns with the ideal network configuration. For example, consumers (C) can contact providers (P) and vice versa. The directional arrows denote the ability of each actor to initiate communication within the system.

3.3 Priority-based Scheduling

An offloading system typically encompasses diverse tasks with varying complexities and priorities, ranging from short to long-term and time-critical to non-time-critical tasks. Accordingly, suboptimal scheduling can result in non-time-critical tasks obstructing the system, leading to delays in the execution of time-critical tasks or missing a deadline. Consequently, deploying a priority-based scheduler to mitigate these circumstances is essential. We describe the fundamentals of our scheduler in the following.

3.3.1 Task Priorities: Tasks are classified into two distinct categories: low-priority and real-time tasks. Low-priority tasks are not time-sensitive and can tolerate execution times of several seconds, however, blocking system resources during their execution. In contrast, real-time tasks have brief execution periods and stringent deadlines that must be met.

3.3.2 Resource Types: Moreover, an offloading system may distribute resources across different locations, varying from geographically nearby to distant ones, leading to variance in latency. Resources connected to the same router are considered the nearest available resources. We refer to edge resources if a resource is located behind an external router, such as residential or corporate resources. If a resource is in local topological proximity, it becomes a favorable offloading target.

3.3.3 Task Scheduling: Our scheduling mechanism utilizes a priority-based approach that applies individual scheduling algorithms based on the priority level of the task and latency in between consumers and providers. Tasks with higher priorities are delegated to the nearest available provider to minimize latency.

4 EVALUATION

This section presents our evaluation including the scheduling strategies and the overall setup. This allows us to empirically measure how simple priority-based scheduling strategies can enhance the service level of latency-sensitive applications in mixed (cloud/edge) environments. We implemented a real-time-capable prototype on top of the Tasklet middleware to conduct the experiment. Additionally, we added different scheduling strategies per priority class utilizing varying device types within the local network, edge, and cloud. To gather realistic data, we performed edge offloading in real-world experiments. Lastly, we discuss our findings and provide open access to our data [1] as a valuable reference for further investigations.

4.1 Scenarios

We evaluate five different scenarios that are summarized in Table 1, providing a comprehensive evaluation of the approach under varying conditions. Per scenario, we captured the timings of the whole Tasklet lifecycle (see Figure 2).

- *S1 Mixed Random:* In this scenario, real-time as well as low-priority tasks are assigned randomly to any available resource. Additionally, all direct communications have been forcefully suppressed, resulting in the need for communication to be relayed. Therefore, this scenario represents the most unfavorable environment for real-time offloading and acts primarily as a baseline for our experiments.
- *S2 Mixed Relayed:* This scenario prefers local over remote resources for real-time tasks. Initially, an effort is made to allocate resources within the local network. If this fails, an edge resource is considered, and when this fails, resources in the cloud are utilized. Whereas low-priority tasks are assigned randomly. All communications with local and edge resources are forced to use the relay.
- *S3 Mixed Direct:* This scenario is similar to *S2* even though that only the communication to the edge resources must be relayed.
- *S4 Separated Relayed:* Within this scenario, real-time tasks are scheduled as in *S2*. Low-priority tasks are only assigned to cloud resources to prevent them from allocating resources with low latencies. The Relay usage is the same as for scenario *S3*.
- *S5 Separated Direct:* In the last scenario, the scheduling of both task types is equal to the one in *S4*. Additionally, direct communications to all resource types are enabled. This approach has the potential to demonstrate a plausible reduction in latency through the utilization of direct communication.

Table 1: Scheduling strategies for real-time and low-priority tasks in different environments. d denotes direct connections, whereby r denotes relayed connections.

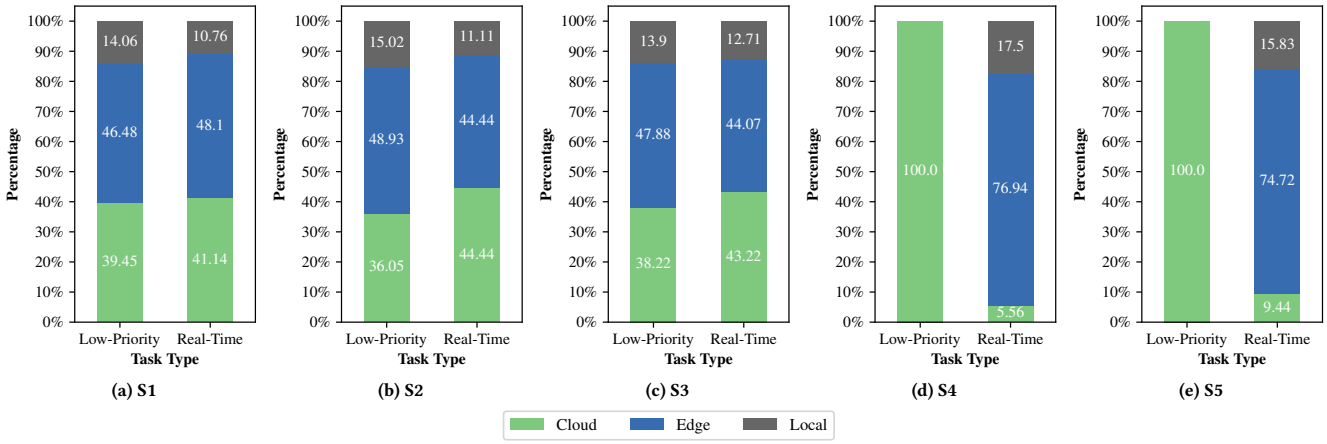
	Scheduling per Priority Class	
	High/Realtime	**Low**
S1	random	random
S2	local_r < edge_r < cloud_d	random
S3	local_d < edge_r < cloud_d	random
S4	local_d < edge_r < cloud_d	cloud_d
S5	local_d < edge_d < cloud_d	cloud_d

4.2 Setup

Our experimental setup consists of a total of 31 providers, 31 consumers, one broker, and one relay server. Four of these are local providers, which are located in the same (local) network as the consumers. These providers represent unused resources within an organizational unit, such as a colleague's computer. The consumer, the broker, and the local providers were located at the University of Hamburg and shared the same router. Each of the devices was connected using a 2 Gigabit Ethernet with an 801.2ad bonding. Fourteen edge devices were distributed in pairs of two across seven different residential homes in Hamburg, Germany. Since we are only interested in the end-to-end network transmission times, we deliberately refrain from naming the hardware specs such as CPU model and RAM. Each edge device was equipped with a 1 Gigabit Ethernet network interface. The cloud resources have been divided into three separate distance classes. In the first class, four cloud resources are located in proximity in Düsseldorf, Germany. The second class consists of eight geographically distant cloud resources situated in Hillsboro, USA. Despite this distance, they are connected by a good peering. Finally, the third class includes a single cloud resource located in Seoul, South Korea, which has considerably

[1] https://github.com/KevinRoebert/Latency-Aware-Edge-Computing

Figure 4: Bar charts showing the (rounded) proportion of tasks across different environments.



(a) S1 (b) S2 (c) S3 (d) S4 (e) S5

poor peering/latency. The relay server was deployed at a cloud server located in Baden-Baden, Germany.

4.3 Results

First, the matchmaking outcomes for our scheduler in the five distinct situations are examined. Subsequently, an in-depth analysis of the network time characteristics exhibited by each resource type during the experimental trials is presented. Finally, we summarize our results and derive conclusions.

In the baseline scenario *S1*, the allocation of low-priority and real-time tasks among all available resources was random. Consequently, an equal proportional distribution with respect to the resource types was observed in Figure 4a. The Figure 4a shows the proportion of tasks across the environments: cloud, edge, and local. Particularly for time-critical tasks, unfavorable scheduling is already apparent here. 41.14% of the critical tasks were scheduled to possibly slow cloud resources. Less critical tasks occupy valuable low-latency resources such as local and edge environments. As a result, only a smaller proportion of these resources is available for time-critical tasks, so they also have to switch to the cloud environment, which may be in an unfavorable location and thus increase the overall latency.

Due to the fact that the tasks are arbitrarily *submitted* to the system, the locality-aware scheduling strategy for the time-critical tasks remains mostly the same in *S2 & S3* (see Figure 4b and 4c). It is thus noticeable that a scheduler must not only distribute the time-critical tasks according to the resource latencies but also ensure that the non-critical tasks do not occupy the low-latency resources.

For the scheduling strategy in scenario *S4 & S5*, it can be seen in Figure 4d and Figure 4e that significantly fewer time-critical tasks reach the cloud environment. Such a strict reservation of local and edge resources for time-critical tasks is not realistic. However, a strict separation into cloud, edge, and local could be replaced in favor of a continuous measurement of latencies and a corresponding classification. This would allow poorly connected edge resources to be valuable targets for non-critical tasks. As a result, resources would be used more effectively in the overall system. It should be noted that the latencies depend on the combination of communication partners. Accordingly, such a strategy would be helpful for recurrent tasks since the scheduler can try to predict submission in this case.

Table 2: Network times of the three cloud environments in ms.

	Loc.	Mean	Std.	Min	25%	75%	Max
cloud	DEU	39.33	19.34	28.0	31.0	39.0	120.0
cloud	USA	201.65	28.54	186.0	191.25	199.25	320.0
cloud	KOR	351.66	58.32	317.0	318.0	369.0	419.0

In the following, the network times describe the sum of all transmission times when offloading a task. This includes requesting a suitable resource from the broker, the subsequent offloading to the provider, and the respective responses.

Table 2 highlights the weaknesses of cloud-only solutions that do not take appropriate measures to mitigate latency. Depending on the location of the cloud services, there are significant differences in network times, even though they typically have strong connectivity. A geographically close cloud would allow 22 FPS with an assumed execution time of 5 ms. This is already below the threshold of 24 FPS for typical video applications. Only 2 FPS (KOR) to 4 FPS (USA) are possible when the cloud is even further away. Thus, our experiment's *mean* network time for the cloud environments is 148.93 ms or 6 FPS as shown in Figure 5. Accordingly, bringing the cloud resources closer to the respective end users would be necessary. This can be difficult for several reasons. Setting up new cloud sites are expensive and not always economical. The existing resources at the edge offer an alternative.

During the experiment, we distinguished between two types of edge resources. In scenarios *S1* to *S4*, these were exclusively accessible via a relay. In scenario *S5*, direct connections were used. Table 3 shows the significant difference between the common methods using a relay and our direct connections. Whereas relayed resources require on average 62.60 ms, the time is almost divided by half to

Table 3: Network times of the local and edge environments in ms. d denotes direct connections, whereby r denotes relayed connections.

	Mean	Std.	Min	25%	75%	Max
local_r	66.49	21.61	50.0	57.75	68.0	183.0
local_d	1.54	1.95	0.0	0.0	2.0	8.0
edge_r	62.60	20.90	42.0	53.0	65.0	178.0
edge_d	34.33	12.99	21.0	27.0	36.0	130.0

34.33 ms by direct connections. This behavior is to be expected, as the relay adds additional hops. If we assume again an execution time of 5 ms, this results in 14 FPS for relayed edge resources and 25 FPS for direct resources. In contrast to the tested cloud resources, this allows us to run real-time applications with high requirements, such as virtual reality, at the edge. Furthermore, we observed that our edge resources have less standard deviation than the closest cloud resources. This is a promising indicator that edge resource latencies can be predicted.

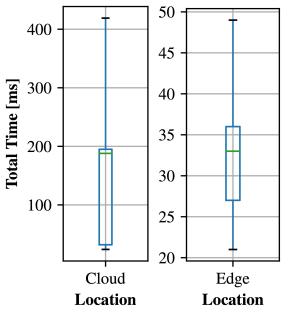


Figure 5: Network times of cloud and edge resources, while both using direct connections.

The importance of a discovery mechanism that can detect the resources available in its network is shown in Table 3. If a system cannot detect that two nodes are located within the same network, this can lead to an increase in delay of 43 times on average. However, this can usually be easily avoided by using NAT loopback support or sending additional local addresses for connection setup attempts.

With the help of the measurements provided, more realistic simulations can be performed based on various input parameters of a scheduler. For example, it can be shown how a system behaves with respect to missed deadlines under different scheduling strategies. This allows optimization for the respective use case and the design of an adapted scheduler. The present study can provide several insights for future schedulers, including but not limited to the following: (I) For effective scheduling, it is recommended that the scheduler allocates nearby resources to tasks with strict and short deadlines while also ensuring the availability of a certain contingent of resources for such tasks. (II) Geographically proximate edge resources can compete with cloud resources and be a worthwhile target for real-time computations. (III) The variance for latency in edge environments can be comparably low or lower than that of geographically proximate cloud environments. Thus, they may be predictable.

5 CONCLUSION

This paper presents a computation offloading system that can utilize edge resources within organizations, public entities, or individual residences to provide real-time services while minimizing latency. We achieved this by abstracting from the heterogeneity of devices at the edge by integrating the Tasklet system and extending it with proactive middlebox traversal techniques resulting in an ideal overlay network with direct connections instead of communication via relays. We provided an implementation showing the feasibility of our approach and conducted empirical experiments. Our results suggest that in contrast to traditional centralized cloud computing models, edge offloading has the potential to decrease latency effectively in real-time applications. Furthermore, the outcomes and insights presented in this paper can serve as a valuable reference for researchers, practitioners, and system designers in developing more authentic simulations that more accurately replicate the performance of edge offloading in real-world situations.

In the future, we also plan to use the collected data as a simulation baseline for optimized scheduling algorithms for real-time and time-critical applications. Furthermore, we want to investigate to what extent we can reduce the latency even further by applying decentralized scheduling strategies and thus avoiding communication with a broker. Moreover, we want to investigate whether pipelining can reduce latencies, especially in the context of data streams.

REFERENCES

[1] International Telecommunication Union. 2003. ITU-T Recommendation G. 114-One-way transmission time. https://www.itu.int/rec/T-REC-G.114-200305-I.

[2] Sebastian Henningsen, Martin Florian, Sebastian Rust, and Björn Scheuermann. 2020. Mapping the interplanetary filesystem. In *IEEE IFIP Networking Conference*.

[3] Mahadev Satyanarayanan. 2017. The emergence of edge computing. *Computer*, 50, 1.

[4] Lorenzo Corneo, Nitinder Mohan, Aleksandr Zavodovski, Walter Wong, Christian Rohner, Per Gunningberg, and Jussi Kangasharju. 2021. (how much) can edge computing change network latency? In *IEEE IFIP Networking Conference*.

[5] Lin Wang, Lei Jiao, Ting He, Jun Li, and Max Mühlhäuser. 2018. Service entity placement for social virtual reality applications in edge computing. In *IEEE Conference on Computer Communications*.

[6] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. 2015. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM transactions on networking*, 24, 5.

[7] Vincenzo Scoca, Atakan Aral, Ivona Brandic, Rocco De Nicola, and Rafael Brundo Uriarte. 2018. Scheduling latency-sensitive applications in edge computing. In *8th International Conference on Cloud Computing and Services Science*.

[8] Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, and Jussi Kangasharju. 2020. Pruning edge research with latency shears. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*.

[9] Heiko Bornholdt, Kevin Röbert, Martin Breitbach, Mathias Fischer, and Janick Edinger. 2023. Measuring the edge: a performance evaluation of edge offloading. In *IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events*.

[10] Dominik Schäfer, Janick Edinger, Justin Mazzola Paluska, Sebastian VanSyckel, and Christian Becker. 2016. Tasklets: "better than best-effort" computing. In *IEEE International Conference on Computer Communication and Networks*.

[11] Dominik Schäfer, Janick Edinger, Sebastian VanSyckel, Justin Mazzola Paluska, and Christian Becker. 2016. Tasklets: overcoming heterogeneity in distributed computing systems. In *IEEE International Conference on Distributed Computing Systems Workshops*.

[12] Heiko Bornholdt, Kevin Röbert, and Mathias Fischer. 2023. Low-latency tls 1.3-aware hole punching. In *IEEE International Conference on Communications*. (in press).

[13] A. Keranen, C. Holmberg, and J. Rosenberg. 2018. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal. RFC 8445.