2022 IEEE International Conference on Edge Computing and Communications (EDGE) | 978-1-6654-8140-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/EDGE55608.2022.00022

uDiscover: User-Driven Service Discovery in Pervasive Edge Computing using NDN

George Torres*, Reza Tourani†, Abderrahmen Mtibaa‡, Diana Stelmakh†,
Satyajayant Misra†, Srikathyayani Srikanteswara§, Yi Zhang§, Sanzida Hoque‡
*New Mexico State University {gtorresz, misra}@cs.nmsu.edu
†Saint Louis University {reza.tourani, diana.stelmakh}@slu.edu
‡University of Missouri–Saint Louis {amtibaa, sanzida.hoque}@umsl.edu
§Intel Labs {srikathyayani.srikanteswara, yi1.zhang}@intel.com

***Abstract*—New breed of applications, such as autonomous driving and their need for computation-aided quick decision making has motivated the delegation of compute-intensive services (*e.g.,* video analytic) to the more powerful surrogate machines at the network edge–*edge computing (EC)*. Recently, the notion of pervasive edge computing (PEC) has emerged, in which users' devices can join the pool of the computing resources that perform edge computing. Inclusion of users' devices increases the computing capability at the edge (adding to the infrastructure servers), but in comparison to the conventional edge ecosystems, it also introduces new challenges, such as service orchestration (*i.e.,* service placement, discovery, and migration). We propose *uDiscover*, a novel user-driven service discovery and utilization framework for the PEC ecosystem. In designing *uDiscover*, we considered the Named-Data Networking architecture for balancing users workloads and reducing user-perceived latency. We propose proactive and reactive service discovery approaches and assess their performance in PEC and infrastructure-only ecosystems. Our simulation results show that *(i)* the PEC ecosystem reduces the user-perceived delays by up to 70%, and *(ii) uDiscover* selects the most suitable server–"accurate" delay estimates with less than 10% error–to execute any given task.**

Keywords: Edge computing, resource discovery, service orchestration, Named-data Networking.

I. Introduction

Over the last few years, there has been an increase in the number of compute-intensive applications, such as Augmented Reality (AR) and autonomous driving, which require ultra-low latency data processing in the proximity of the users. The execution of such applications, often, exceed the processing capability of standalone devices such as smartphones, or tends to exhibit significant battery draw. This has led to the use of edge computing, where computing resources are deployed physically close to end-users. However, the forecast of increasing volume of services and data, will require an all-hands-on-deck approach [1]. This has motivated the vision of the Pervasive Edge Computing (PEC) ecosystem [1], which can utilize the devices at the network periphery, such as mobile and client devices, third party servers etc., to dynamically participate in the compute pool to improve service availability and delivery.

In this paper, we argue that the evolving Named Data Networking (NDN) architecture is a much better choice for meeting the unique needs of edge computing applications. In contrast to the IP-based networking architectures, NDN uses the application-defined names at the network layer for communication, while it features in-network caching and built-in security. We aim to leverage NDN to design the *uDiscover* framework to perform resilient edge service discovery (resulting in service utilization) by enabling resources (storage, computation) to be shared through names, offering opportunities for "plug-n-play" service provisioning at the network edge.

What to Discover? Applications need to discover the following information: (*i*) the availability of services, and (*ii*) the resources (*e.g.,* capabilities and load) of each offering server. The first piece of information is needed so that users can discover the service(s) providing the functionality they need along with service-related metadata (*e.g.,* service description). The second one helps estimate the required time to perform a given user task if a particular server is chosen.

Prior research in the area of NDN edge computing has been extensively focused on network-driven approaches [2], [3] or proxy-driven approaches [4]. These approaches are limited to discovering only upstream (infrastructure) servers and are oblivious to any peer server, *i.e.,* devices within one hop from the users such as wireless user devices. In addition, these approaches rely on servers' utilization as the sole metric in choosing the server without accounting for the communication delay due to the transferring of input/output data, which can be very large for applications such as video annotation. *uDiscover* employs task profiling at the servers to estimate task computation and communication delays estimation, which users utilize for making informed server selection decisions.

Our novel **contributions** include: *(i)* make the case for the benefits of leveraging user devices (uServers) to supplement existing infrastructure computing resources (iServers); *(ii)* design and implement *uDiscover*, a user-driven framework (both proactive and reactive) that makes informed server selection decision based on users' task profiles and available resources at the network edge; and *(iii)* perform an extensive set of simulations to quantify the gains of *uDiscover* when compared to infrastructure-only edge computing (iServer-only).

II. Background and Related Work

A. Name Data Networking

The Named-Data Networking (NDN) architecture [5], [6] shift the existing host-centric IP architecture to a data-centric

2767-9918/22/$31.00 ©2022 IEEE
DOI 10.1109/EDGE55608.2022.00022

paradigm, in which any network entity with a cached copy of the data can satisfy users' requests. In NDN's receiver-driven (*i.e.,* pull-based) communication model, a user's application sends a request, *i.e., Interest packet*, into the network including the name of the requested data. On receiving the Interest, the producer's application sends back the requested data, *i.e., Data chunk*, consisting the data and its unique name that are bound together via producer's signature–enabling data integrity and authenticity verification.

NDN's unique content naming and strategy layer allow packet processing at the network layer and flexible packet forwarding. Moreover, NDN reduces redundant content delivery using its stateful forwarding plane. To realize the stateful forwarding plane, routers are equipped with three data structures. The Forwarding Information Base (FIB) contains name prefixes along with pertinent outgoing interfaces, which is used for Interest forwarding. The Pending Interest Table (PIT) keeps track of on-the-fly Interests and enables Interest aggregation. The Content Store (CS) enables all the network entities, *i.e.,* routers, to perform in-network content caching.

B. Edge Computing in NDN

The advances in edge computing (EC) has primarily targeted areas, such as optimizing resource management, energy consumption, and task scheduling [7]–[10]. However, EC's networking aspects, such as adaptive task forwarding or seamless resource discovery in highly dynamic and heterogeneous edge environments, have not received much attention.

A few initiatives have explored the suitability of the ICN paradigm for EC and concluded that such ecosystems greatly benefit from the distributed nature of NDN [1], [11], [12]. Named-Function Networking (NFN) [13] was the first detailed attempt in this area with a detailed focus on function placement and execution. NFN's fundamental idea was to leverage the unique function naming feature of NDN for locating the computing resources. NFaaS [14] extended NFN's vision to a function-as-a-service platform, enabling the users to request function executions without system provisioning. However, these proposals fall short in executing resource-intensive functions due to their simplistic design that needed the network to maintain users' and edge servers' states during function execution. RICE [2] and ICedge [3] addressed this shortcoming by decoupling service discovery and service invocation; users will receive the instruction of how and when to request the result of the service execution. Other initiatives have attempted to optimize network-assisted resource discovery and deploying edge-centric security measures [4], [15], [16].

Different from these efforts, we aim at assessing the advantages of a user-driven resource discovery scheme in the dynamic PEC ecosystem [1], which requires more complex and sophisticated service orchestration operations.

III. Pervasive Edge Ecosystem: System Model

We consider a computing ecosystem that comprises the Cloud and the pervasive edge computing (PEC) ecosystem [1]. The PEC ecosystem is envisioned as the edge-to-Cloud continuum, including all the computing resources from the constrained Internet of Things (IoT) devices and users' personal devices to multi-access edge computing (MEC) [17], and fog computing [18]. PEC augments the existing pre-deployed computing infrastructure with the users' computation resources–creating a larger and more heterogeneous resource pool for execution of requested services in users' proximity. Thus, enabling markedly better performance and democratizing the execution of services at the network edge.

We consider two types of PEC computing resources (PEC servers), namely *infrastructure servers (iServers)* that are dedicated infrastructure servers and *user servers (uServers)* which are users' standalone computing resources like personal computers. The iServers are the static computing resources while uServers are more dynamic in nature as they can join or leave the network at will. We consider a heterogeneous PEC, in which the iServers are computationally more capable than the uServers. Fig. 1 depicts the high level system consisting of a user, Cloud, iServers, and uServers. We assume that uServers are connected to infrastructure, *i.e.,* base stations, and are accessible using additional Device-to-Device (D2D) wireless technologies, such as WiFi Direct, Bluetooth, or Zigbee. In this paper, we assume all entities communicate using NDN.

Each service provider owns a (set) of service(s), such as image annotation and video super-resolution. While the service providers have their computing infrastructure (*i.e.,* owned or leased from the Cloud), we consider that they also offload the execution of the requested services to the iServers and uServers. We assume the existence of appropriate security and access control mechanisms, which allows the service providers to onboard PEC servers–enabling computation offloading to iServers and uServers. Considering the large number of services, we assume that each service is offered by only a subset of the servers; requiring service/resource discovery. A user is typically a service requester. However, considering the highly dynamic nature of the PEC ecosystem, a user may become an uServer by joining the resource pool and dedicating their resources for execution of other users' services. Services can be either static (*e.g.,* static videos or web content) or dynamic (*e.g.,* annotation of videos or images). In this paper, we focus on dynamic services that require input data from service

Fig. 1. The pervasive edge computing ecosystem, including the pre-deployed edge servers (iServers) and the users' devices (uServers). PEC allows the users to directly discover the nearby uServers without relying on the base station.

requesters (*e.g.,* a user's image/video for the annotation) or other service providers (*e.g.,* live video super-resolution).

IV. User-driven Service Discovery

Contrary to the existing network-driven service discovery [2], [3], in *uDiscover*, users make offloading decisions using a utility function, based on task profile and per-server estimated latency using a utility function, gathered from the servers with the objective of selecting the best server.

A. Utility Function

Task scheduling can be formulated as an optimization problem, which users can solve to sort the PEC servers (uServers and iServers) for any given task at a time t. In this way, users can forward requests to the most suitable server. We consider two main characteristics of services/tasks, namely, computation complexity and the input data size. Using these characteristics, one can model services spanning from heavy compute and low data input such as video annotations to low compute and heavy data such as motion detection.

Users will discover all available servers s_i, $i = 1 \cdots N$ and estimates at a given time t the compute delay, $d_{comp}(S_t, i)$, and the communication delay, $d_{comm}(S_t, i)$, for any service S_t to run on server s_i. These estimates are measured based on information shared by servers and gathered during the discovery process. Any server s_i will, periodically (*i.e.,* proactive) or on-demand (*i.e.,* reactive), send its current utilization, u_i (*e.g.,* CPU usage ratio, waiting queues), its capacity, c_i (*e.g.,* infrastructure servers have larger capacity than the smaller PEC servers), as well as the average communication delay, *i.e.,* d_i, measured using the last K tasks received by the server.

As a user sends a service discovery and gathers all servers' information, it estimates the end-to-end delays to run its tasks on all servers, and sorts them based on these delay estimates:

$$d_{comp}(S_t, i) = BaseC(S_t) \times \frac{u_i}{c_i}, \tag{1}$$

where $BaseC(S_t)$ is the optimal computation unit for task S_t (*i.e.,* with absence of queuing and concurrent computation).

$$d_{comm}(S_t, i) = d_i \tag{2}$$

In this paper, we used $c_i = 1$ for iServers and $c_i = 1/3$ for uServer. Thus, estimating latency for the service S_t is measured as:

$$\min_i \left(d_{comp}(S_t, i) + d_{comm}(S_t, i) \right), \tag{3}$$

where the best server to execute the task, s_i, is the one that minimizes the function by Eq. 3. These estimates are based on information gathered from each server. We have proposed and tested a proactive and reactive user-driven approaches to discover and gather resource updates from the edge servers.

B. Proactive Utilization Updates

In NDN, any data content must be requested using interest packets. Thus, each infrastructure entry point (*e.g.,* access point or base station), which we refer to as BS, probes the servers in its vicinity (*i.e.,* limited to k-hops away from the BS), periodically every period Δ. These interests are multicasted periodically to all servers. To limit the overhead caused by the periodic updates, BSs perform scoped flooding, where each update is propagated for a limited number of hops in the network, before it gets discarded.

Servers receiving these interests, reply with data packets that contain the estimated communication delays, the current utilization, and the capacity of the server. The estimates of communication delays is calculated based on averaging the last T tasks from each service profile; the communication delays would be similar since most of the users are within the k NDN hops radius. The BS gathers all responses and aggregate them into one resource discovery manifest, which will be sent to any users requesting a service from the BS.

Users, with tasks to execute, first send a discovery interest as a one-hop broadcast message on all available interfaces. All uServers receiving the discovery interest reply with their utilization, communication delays, and capacity. BSs receiving the discovery request, reply with their aggregated resource discovery manifest. Upon receiving all data, users will compare the delays to execute its task on all servers (*i.e.,* based on Eq. 3), and select the server that minimizes the end-to-end latency. Note that, the utility function deployed in this paper can be easily changed to a multi-objective function which takes into consideration load balancing, overhead, cost, etc.

Fig. 2 depicts the sequence diagram of our user-driven approach. Blue lines represent the proactive approach messaging the BS periodically (every Δ seconds) with multicast message to proactively gather and store the resource manifest that aggregates all resource information from available iServers within a 3-hop radius and direct uServers. When users broadcast a discovery interest, they will instantly receive a data content from BS (from the cached resource manifest) as well as the uServers accessible via other wireless interfaces such as Bluetooth, Zigbee, WiFi-Direct, etc.). In this example the user selected server (s_i), and sent the task to the server. Server reserves resources, updates its utilization, and requests the input data from the user (if any) to run the task. Upon receiving the data, the server executes the task; then waits for the request to send the task execution results and frees the resources (*i.e.,* reduce its utilization ratio).

C. Reactive Utilization Updates

The reactive approach does not require periodic exchange of updates. Similar to the proactive approach, users broadcast (*i.e.,* 1-hop broadcast) a discovery request on all its interfaces. All uServers reply with their utilization, capacity, and communication delays. However, BSs receiving the discovery interest, reacts by initiating an on-demand discovery of services within a k-hop radius. The process is similar to the proactive approach, however it would be triggered on-demand (instead of periodically) as the BS receives a request from the user.

The reactive approach messaging is depicted in Fig. 2 in red. BSs receiving the discovery interest from the users will not respond instantly, as per the proactive approach, however BSs send service discovery interests to gather all resource

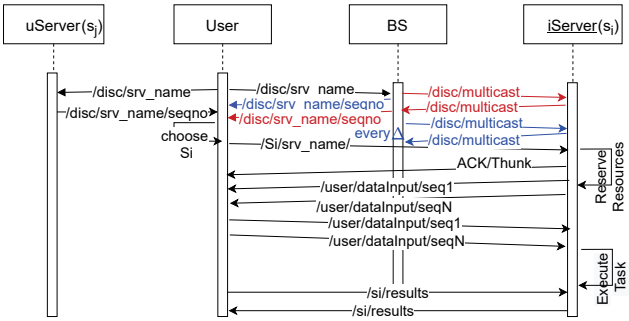


Fig. 2. *uDiscover* interactions across all parties; Blue and Red represent the message exchanges of the proactive and reactive approaches, respectively.

information and relay them back to the requesting user. This on-demand procedure does not require caching of manifest nor sending periodic messages when there are no users requesting task executions. However, in order to control the overhead of this reactive approach, we have implemented a freshness period (this period is set to 100ms) to cache the results and avoid sending "unecessary" discovery messages for requests received within this period.

V. Evaluation

In this section, we evaluate, compare, and discuss the performance of our *uDiscover* approaches to an iServer-only approach, which proactively ($\Delta = 1$) discovers and selects only iServers–we will quantitatively measure the benefits of leveraging user' devices, uServers, in a PEC setting.

A. Experimental Setup

We used ndnSIM [19], a module of ns-3, to implement and evaluate *uDiscover*. We ran our simulation on a Desktop class machine with a Intel 16 core CPU and 64 GB memory. We implemented three applications to enable the interactions between the users, base stations, iServers, and uServers.

Implementation Scope: In the user discovery application, each user initiates the resource/service discovery protocol and collects the available resources either from the corresponding base station or directly from the nearby uServers. Upon choosing the best server, the user informs the selected server to fetch the data from the user. The resource probing application enables the base stations to elicit the available resources of the proximal servers by sending hop-limited probing messages to the nearby servers. In the reactive approach, each base station probes the nearby resources only on receiving the users' discovery requests while in the proactive approach, the probing takes place periodically. Finally, in the server application, the server designated to execute the service initiates the communication with the user to retrieve the data needed for the requested service, executes the requested service, and returns the result of the service execution to the user.

Network Topology: We created a pervasive edge network topology of 191 nodes, including 126 users, 38 uServers, 9 iServers, and 8 base stations. Each base station serves roughly 20 users and uServers in total. The users and uServers are directly connected to their associated base stations over 1-hop connections while the iServers are connected to the base stations over 2-hops or 3-hops connections. Thus, resulting in the total user-to-uServer distance of one to two hops and the total user-to-iServer distance of roughly three to four hops (*i.e.,* $k \in \{3, 4\}$).

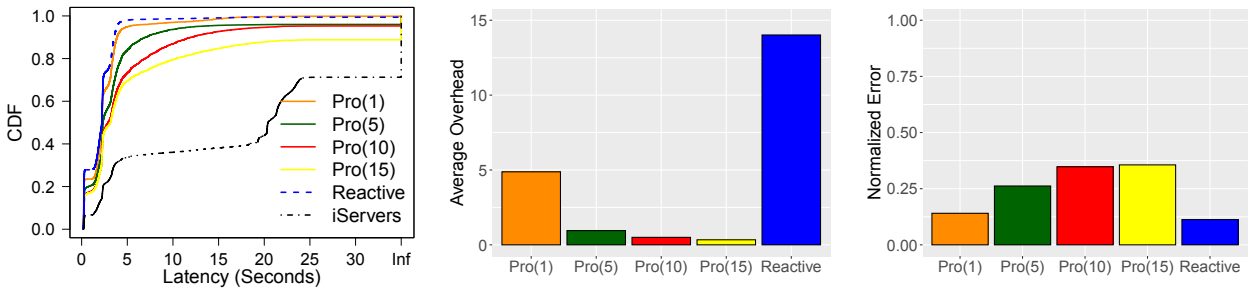
TABLE I
Evaluation setup; Numbers in **bold** are nominal (default) values used when the value of a parameter is fixed.

Parameter	Value Range
Discovery Scenarios	
Proactive BS probing period (Δ)	{1, **5**, 10, 15} seconds
Reactive freshness period	100 ms
Reactive BS probing wait period	40 ms
User service request rate	1 request per {1, **3**,10} seconds
Servers Specifications	
uServers status change intervals	{30 seconds, **100 seconds**, off}
Probability of uServers status update	0.65/0.35 to keep/switch status
Initial utilization of iServers	$\mathcal{N}(10, 10)$
Utilization increment of iServers	$\mathcal{N}(5, 5)$
Initial utilization of uServers	$\mathcal{N}(20, 20)$
Utilization increment of uServers	$\mathcal{N}(15, 10)$
Topology Specifications	
User-uServer link delay	10ms
User-BS link delay	10ms
User-uServer link bandwidth	500Mbps
User-BS link bandwidth	5Mbps
Hops between users and iServers (k)	[3-4]
Average user node degree	[2-4]
Number of iServers	[10-15]

We modeled the connection between the users and base stations as well as the connections between the uServers and base stations as Ethernet peer-to-peer links. Considering the focus of this work on analyzing user-driven resource discovery, we intentionally avoided wireless links to prevent the interference of the wireless channels from impacting our evaluation. We set the bandwidth and delay of the links between the users and base stations and the links between the iServers and the base stations as $500Mbps$ and $10ms$, respectively. We set the bandwidth and delay of the links between the user and uServer as $5Mbps$ and $1ms$, respectively.

We devised a probabilistic model for the uServers to mimic a dynamic PEC ecosystem, in which the uServers can join or leave the pool of computing resource at will. In our model, uServers can have two states, online or offline. To model network dynamicity, each uServer individually decides to continue its current status–remains online or remain offline–or switch its current status–comes online if it was offline or vice versa (refer to Table I for detailed parameters).

Service Configuration: In the PEC, executing dynamic services often require input data. The size of the input/output data and amount of computational resources for performing a given service, however, are service-specific parameters. Considering our simulation environment, we modeled four representative services with diverse communication and computation characteristics. For each service, we defined the input and output data sizes in terms of the number of packets (1 KB sized packet) and the service execution according to Table II.



(a) User perceived latency (b) Averaged overhead (c) Normalized error

Fig. 3. Comparing (a) user perceived latency (Inf denotes infinite delays resulting from data/task loss), (b) overhead, and (c) error of reactive and proactive (with different update periods, Δ) approaches; user service request rate and join interval are respectively fixed to 1pkt every 3 seconds and 100 seconds.

B. Evaluation Metrics

All metrics shown in this subsection are measured as an average of five simulation runs. Each simulation runs for 360 seconds. We consider the following evaluation metrics:

1) *User perceived latency*: the time elapsed from the user sending the resource discovery request until the user receiving the result of the offloaded service, including service discovery, invocation, and execution.
2) *Average overhead*: the overhead is the additional number of packets transmitted per service discovery transmitted between the base stations and the proximal servers, averaged over all users. The average overhead is measured as the ratio between the overhead of any approach with respect to the iServer only overhead.
3) *Normalized Error*: the difference between the estimated end-to-end latency for task execution and the user perceived latency normalized by the perceived latency.

C. Results and Analysis

1) Impact of Update Periods: We first compare iServer-only and *uDiscover*'s reactive and proactive (with varying update periods Δ) approaches in terms of latency, overhead, and error (Fig. 3). As shown in Fig. 3(a), the cumulative distribution function (CDF) of the user perceived latency in *uDiscover*'s reactive and proactive approaches (with the higher update frequency) outperform the iServers only approach in selecting the best servers, resulting on lower latencies. The difference gap is further amplified in longer delays where the reactive approach and the proactive approach with the highest update frequency (*i.e.,* Pro(1) where $\Delta = 1s$) outperform all other approaches. 60% of Pro(1) and reactive approaches' tasks completed in less than 2 seconds compared to 3 to 20 seconds for the other approaches. For the proactive approach, reducing the update frequency (*i.e.,* increase Δ values) results in considerable user perceived delay increase; up to 20%. The reason is for the larger Δ values, users will use stale information that can results is choosing overloaded servers. Finally, we show that the iServers only approach performs poorly when compared to reactive and proactive approaches with larger Δ values. Servers become overloaded and fail to schedule tasks for execution. Delay performance is correlated with the efficacy of the delay estimation of each approach as shown in Fig. 3(c). High frequency updates (*e.g.,* reactive and proactive with $\Delta = 1s$) uses fresh(er) data, thus making the least estimation errors resulting in better server selection and overall performance.

TABLE II
SERVICE CONFIGURATION

Data Input \ Exec Time	1s [$\mathcal{N}(1, 0.03)$]	0.1s [$\mathcal{N}(0.1, 0.01)$]
1000 pkts [$\mathcal{N}(1000, 0.01)$]	Service 1	Service 4
10 pkts [$\mathcal{N}(10, 0.1)$]	Service 2	Service 3

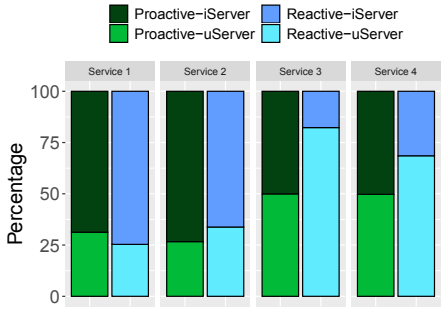


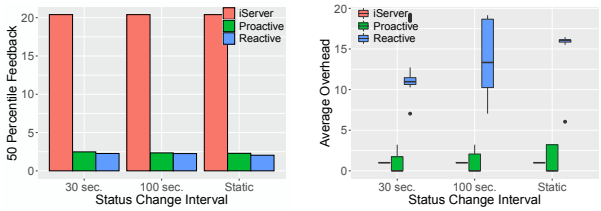
Fig. 4. Distribution of services across uServers and iServers.

The good latency performance of the reactive approach, however, comes at the cost of a larger overhead (refer to Fig. 3(b)). The reactive approach averages $14times$ the overhead of iServer-only. However the proactive approaches achieve better latency performance while keeping overhead low (*e.g.,* 30% of the iServer-only overhead for Pro(10) approach). Thus, when comparing both *uDiscover* approaches, the proactive approaches tend to achieve a better latency-overhead trade-off; for instance proactive ($\Delta = 5s$) has 3-100 $\times$ less overhead, but only 1 to 23% longer delays (measured as % of tasks having delays within 10s).

We also measure the distribution of tasks across different servers (*i.e.,* uServer and iServer) as shown in Fig. 4. Services 1-4 have different input and computation loads. We show that services with high computation load (*e.g.,* service 2) tend to be forwarded more towards iServer while services with high communication load (*e.g.,* service 4) tend to be transferred more towards uServers. Both approaches are following this trend, with the reactive approach selecting 10% to 20% more uServer when needed, resulting in shorter delays and better

(a) Latency vs. Request Rate (b) Overhead vs. Request Rate

Fig. 5. The impacts of service request rate on user perceived latency and communication overhead.



(a) Latency vs. uServers update (b) Overhead vs. uServers update

Fig. 6. The impacts of uServer dynamicity on user perceived latency and communication overhead.

load balancing.

2) Impact of User Service Request Rate: We studied the impact of service traffic (user service request rate) on the latency and overhead. We show, in Fig. 5(a), that as the user request increases, the network become saturated and iServer-only fails to execute more than 50% for 1 request/s scenario and *uDiscover*'s approaches help execute tasks in less than 10× the delays achieved by iServer-only for the 1 request/3s scenario. While the reactive approach performance comes at the cost of 2-30× the overhead of iServer-only. Pro(5) achieves better overhead and delay performances. Note that iServer-only probes discover resources every 1 second, thus higher overhead, but fails to achieve better user perceived delay as it doesn't leverage all resources in the PEC setting.

3) Impact of uServers Dynamicity: We further studied the impact of PEC's dynamic ecosystem, in which the uServers join and leave the network, on user perceived latency and communication overhead. Fig. 6(a) compares the average delays when the uServers dynamicity decreases. We show that our approaches, proactive and reactive, are resilient to uServers dynamicity (Fig. 6). The approaches manage to overcome disruption and select uServers or iServers that help reduce the user perceived latency.

VI. Conclusion

We have proposed a new user-driven service and resource discovery for pervasive edge computing networks. Our approach, *uDiscover*, uses information about the resource utilization of infrastructure servers (iServers) and ad-hoc user devices (uServers), in addition to an estimation of the communication delay to make informed decisions on which server can help execute a given task in the shortest delay.

Acknowledgments

This work was partially supported by US NSF awards #2148358, #2028797, #1914635, and #OIA-1757207 and Intel Labs.

References

[1] R. Tourani, S. Srikanteswara, S. Misra, R. Chow, L. Yang, X. Liu, and Y. Zhang, "Democratizing the edge: A pervasive edge computing framework," *arXiv preprint arXiv:2007.00641*, 2020.

[2] M. Król, K. Habak, D. Oran, D. Kutscher, and I. Psaras, "Rice: Remote method invocation in icn," in *Proceedings of the ACM Conference on Information-Centric Networking*, 2018, pp. 1–11.

[3] S. Mastorakis, A. Mtibaa, J. Lee, and S. Misra, "Icedge: When edge computing meets information-centric networking," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4203–4217, 2020.

[4] R. Pirmagomedov, S. Srikanteswara, D. Moltchanov, G. Arrobo, Y. Zhang, N. Himayat, and Y. Koucheryavy, "Augmented computing at the edge using named data networking," in *2020 IEEE Globecom Workshops (GC Wkshps*. IEEE, 2020, pp. 1–6.

[5] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[6] L. Zhang, *et al.*, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[7] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[8] K. Habak, E. W. Zegura, M. Ammar, and K. A. Harras, "Workload management for dynamic mobile device clusters in edge femtoclouds," in *Proceedings of ACM/IEEE symposium on edge computing*, 2017, pp. 1–14.

[9] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, 2011, pp. 301–314.

[10] A. Mtibaa, K. A. Harras, K. Habak, M. Ammar, and E. W. Zegura, "Towards mobile opportunistic computing," in *IEEE 8th International Conference on Cloud Computing*. IEEE, 2015, pp. 1111–1114.

[11] D. Grewe, M. Wagner, M. Arumaithurai, I. Psaras, and D. Kutscher, "Information-centric mobile edge computing for connected vehicle environments: Challenges and research directions," in *Proceedings of the Workshop on Mobile Edge Communications*, 2017, pp. 7–12.

[12] A. Mtibaa *et al.*, "Towards edge computing over named data networking," in *2018 IEEE International Conference on Edge Computing (EDGE)*. IEEE, 2018, pp. 117–120.

[13] M. Sifalakis, B. Kohler, C. Scherb, and C. Tschudin, "An information centric network for computing the distribution of computations," in *Proceedings of the 1st international conference on Information-centric networking*. ACM, 2014, pp. 137–146.

[14] M. Król and I. Psaras, "Nfaas: named function as a service," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*, 2017, pp. 134–144.

[15] A. Mtibaa, R. Tourani, S. Misra, J. Burke, and L. Zhang, "Towards edge computing over named-data networking," in *2018 IEEE International Conference on Edge Computing*. IEEE, 2018, pp. 117–120.

[16] R. Tourani, A. Bos, S. Misra, and F. Esposito, "Towards security-as-a-service in multi-access edge," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 358–363.

[17] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[18] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 2012, pp. 13–16.

[19] S. Mastorakis *et al.*, "On the evolution of ndnsim: An open-source simulator for ndn experimentation," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, pp. 19–33, 2017.