ExEC: Elastic Extensible Edge Cloud

Aleksandr Zavodovski
University of Helsinki, Finland

Nitinder Mohan
University of Helsinki, Finland

Suzan Bayhan
TU Berlin, Germany

Walter Wong
University of Helsinki, Finland

Jussi Kangasharju
University of Helsinki, Finland

Abstract

Edge computing (EC) extends the centralized cloud computing paradigm by bringing computation into close proximity to the end-users, to the edge of the network, and is a key enabler for applications requiring low latency such as augmented reality or content delivery. To make EC pervasive, the following challenges must be tackled: how to satisfy the growing demand for edge computing facilities, how to discover the nearby edge servers, and how to securely access them? In this paper, we present *ExEC*, an open framework where edge providers can offer their capacity and be discovered by application providers and end-users. *ExEC* aims at the unification of interaction between edge and cloud providers so that cloud providers can utilize services of third-party edge providers, and any willing entity can easily become an edge provider. In *ExEC*, the unfolding of initially cloud-deployed application towards edge happens without administrative intervention, since *ExEC* discovers available edge providers on the fly and monitors incoming end-user traffic, determining the near-optimal placement of edge services. *ExEC* is a set of loosely coupled components and common practices, allowing for custom implementations needed to embrace the diverse needs of specific EC scenarios. *ExEC* leverages only existing protocols and requires no modifications to the deployed infrastructure. Using real-world topology data and experiments on cloud platforms, we demonstrate the feasibility of *ExEC* and present results on its expected performance.

1 Introduction

The introduction of cloudlets [33] and fog [11] served as a starting point for the recent developments in edge computing (EC), pushing it as a key enabler for technologies such as Internet-of-Things (IoT), and augmented reality. By bringing the edge servers near end-users, novel applications and services requiring very low latencies have become feasible, and the growing popularity of such applications is pushing the development of EC further.

The benefits of EC are well understood, and some key challenges such as edge server placement have been addressed, e.g. [13]. In most of the previous work, the cloud provider is assumed to have full control over the edge servers. However, we envision a more realistic setting in which any entity can offer their computation capacities for running edge applications and services, and become what we call an *independent edge provider* (IEP). Indeed, telecom operators are actively planning for deployment of Multi-access Edge Computing (MEC) [16] infrastructure. Also, new crowdsourcing platforms have emerged, e.g., [19, 22], having most of their resources at the edge of the network, exactly where the new EC applications would require them. Furthermore, [18, 28] suggest unleashing underutilized computational capacity of smart devices to be used for EC purposes. In all previous cases, the cloud provider will be agnostic of available IEPs and needs a specific means for symbiotic interaction with them. Such interaction consists of i) discovery of IEPs ii) negotiation with IEPs on the deployment of services iii) entering into a contractual agreement with IEPs iv) securely deploying and running the edge services.

In this paper, we propose *ExEC*, *Elastic Extensible Edge Cloud* platform to handle the set of tasks presented above, emphasizing the discovery aspect. To illustrate the motivation behind *ExEC* with an example, we assume the following scenario. An application provider deploys an application composed of both back-end and front-end services (edge) to some cloud facility located in the Northern US. As the popularity of the service grows, user crowds from the other half of the earth, e.g., Southeast Asia, subscribe to the application. The cloud provider offers edge facilities in the US, but not in Southeast Asia. Attempts of administrative personnel to find local IEPs are slow and inefficient. Since resulting QoS is low, the application provider stands at a major risk of losing this new audience.

In the above scenario, *ExEC* would detect the emergent flows of end-user requests, initiate on-path discovery of IEPs, and migrate edge services closer to the end-users, keeping them satisfied and saving the core network's capacity. In dynamic settings where end-users' behavior changes frequently and their locality is hard to predict, *ExEC* has an advantage over interception [24] since it might not be possible to foresee all regions where the application will be used. Moreover, as end-to-end encryption becomes prevalent on the Internet, solutions clinging to the on-path interception of requests are not feasible, mandating the end-to-end approach.

We believe that the open approach enabled by *ExEC* is key to ensuring the take-up of edge computing and making it ubiquitous and pervasive. Unifying the discovery and access of current edge facilities, which might be set up by either cloud providers (e.g., Azure Stack or Amazon CloudFront) or telecom operators (MEC), *ExEC* goes further and opens an opportunity for crowdsourced solutions, combating the "Internet feudalism", brought up in [25]. The advantage of *ExEC* is the capability to be incrementally adopted, and while some IEPs might use it only for the discovery of their facility, the others may utilize automatic migration of services and contractual agreement. To the best of our knowledge, *ExEC* is the first solution to enable the discovery of edge resources without special controllers or specific entry points since *ExEC* advocates using DNS as a global registry of IEPs. All functions of *ExEC* are designed to utilize only existing protocols and do not require any changes to available infrastructure or clients/servers. *ExEC* can coexist with

to be presented at EdgeSys 2019

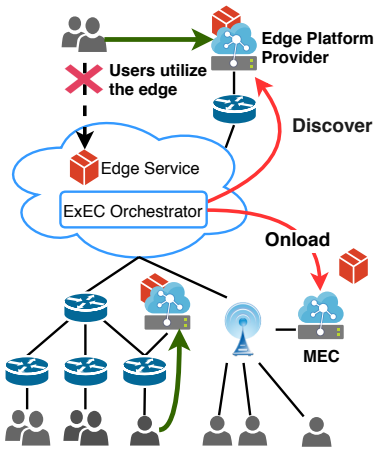


Figure 1. The operation of *ExEC*. The system discovers IEPs on the paths to clients and pushes edge services closer to their consumers.

other approaches, such as pre-deployed edge configuration, enhancing the system performance and reducing administrative overhead.

Using real-world topology data and experiments on cloud platforms, we demonstrate that *ExEC* is feasible and it meets its goals with sufficient performance. We believe that *ExEC* can boost EC by i) providing unified discovery and access mechanism of edge facilities ii) improving the availability of edge facilities by allowing third parties to start offering their computational capacity for edge iii) automatically unfolding the applications towards edge iv) advancing sustainable development of EC by utilization of already existing computing facilities for the purposes of EC.

This paper is organized as follows. Section 2 describes the design and enablers of *ExEC*. Preliminary evaluation results can be found in Section 3. Section 4 discusses *ExEC*'s applicability to various use cases. Related work is the topic of Section 5. Finally, Section 6 contributes on future directions and concludes the paper.

2 System Overview

We illustrate the operation of *ExEC* in Figure 1. There are three main phases: discovery of IEPs, negotiation followed by contractual agreement, and migration of services across administrative boundaries. As it was mentioned, the counter-parties may use *ExEC* either to accomplish all these steps or just select relevant ones, e.g., if the list of the contracted IEPs is known in advance, only dynamic service migrations might be used. Alternatively, if IEP discovered by *ExEC* prefers custom contractual agreement, counter-parties may handle this out of *ExEC* scope. The main entities of *ExEC* are i) all kinds of IEPs, including cloudlets, MEC or crowdsourced edge servers ii) an *application* which is composed of several services, some of which need to be deployed at the edge for optimal performance iii) the *cloud* which hosts the application initially[1] iv) *edge orchestrator* running in cloud which is responsible for executing main functionalities of *ExEC*, such as discovery of IEPs, analysis of request patterns, and dynamic onloading of services to edge.

2.1 Enabling Technologies

At the core of *ExEC* is a capability to discover IEPs located on the paths from the end-users to cloud. We suggest that IEPs should register themselves in the DNS system by adding appropriate SRV records. Such DNS records may cover a particular edge server or a large IEP site. Opportunity to be discovered can bring new customers and can increase revenue, therefore, acting as an economic driver. An ability to gain revenue requires some form of contractual agreement, and this is the next layer of *ExEC*. In common, to seal an agreement between two (or even more) parties we need a third trusted party. Recently, a technology aiming for such a role became widespread: namely, *smart contracts* popularized by the Ethereum [4] platform. Basically, a smart contract is a program wherein correctness of execution is verified by a majority of participants thereby achieving distributed consensus. Due to smart contracts, a centralized trusted party can be replaced by peer-to-peer network; and as long as the majority of peers are fair in the system, the correctness of smart contract execution can be guaranteed. Smart contracts appear as a viable solution for achieving agreement on the scale of the Internet. The feature making them especially attractive is an escrow service they provide out-of-box. As Wright et al. [38] describe, a smart contract can hold the payment of a buyer and transfer it to the seller only after verifying that service was delivered as promised. We conveniently employ smart contracts in the context of *ExEC* as a decentralized solution to handle payments and concordance. We discuss the details of smart contracts utilization in the context of *ExEC* in Section 2.3.

For virtualization and service migration, we rely on existing technologies, such as Docker [3] or Virtual Machines (VMs). Docker containers are capable of live migration [39] and impose low overhead also in other scenarios [26]. Similar solutions exist for VMs [14]. Internally, IEP can rely on, e.g., Kubernetes [5], OpenStack [6] or Docker swarm.

The last layer in *ExEC*'s enabling technology stack is Trusted Execution Environments (TEEs) [31]. With TEEs, even user having root privileges cannot access a protected memory region of an application. TEEs are supported by all major processor manufacturers, most of the attention gained Intel SGX (Software Guard Extensions) [23], while ARM has developed its own TrustZone [8] technology, which was also recently incorporated by AMD [7]. Arnautov et al. [9] show how to protect Docker container with SGX. TEEs are essential for *ExEC* operation only in those cases where edge service handles private or sensitive data; otherwise, *ExEC* does not require TEE.

2.2 Discovery of IEPs

ExEC builds its own view of a network which is centered at the location of the edge orchestrator and comprised of paths to clients together with IEPs discovered along those paths. We refer to this view as *topology* and break down the process of building such a topology into three steps. Figure 2 provides an intuitive overview, and we give a detailed description of each step below.

Step 1: Determine paths to clients: Edge orchestrator monitors incoming requests, and for every new request from a subnet that orchestrator has not previously seen, the orchestrator initiates network tomography procedure. In its purest form, this may be just sending out a `traceroute` request to client's address, which is enough to identify the path from the cloud to the client and gather information about latencies between on-path routers. *ExEC* is not limited to usage of `traceroute` tool, which can be either completely replaced or augmented by crowdsourced methodologies, active network sniffing such as `nmap`, precomputed network topology

[1] Some of the application's services might also be initially predeployed to edge.

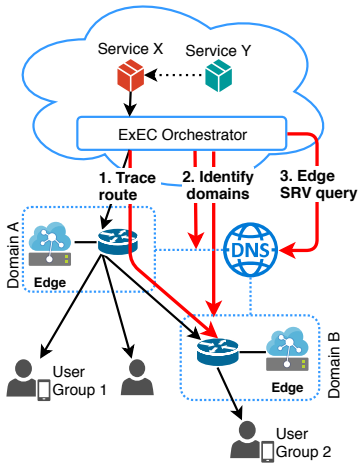


Figure 2. Discovery of IEPs in *ExEC* works by tracing paths to clients (1), identifying on-path domains (2), and querying DNS servers of discovered domains for edge SRV records (3).

maps, and other methods. For now, we assume `traceroute` to be sufficient for the functionality of *ExEC*.

Step 2: Determine on-path DNS zones: The network tomography performed in the previous step gives *ExEC* the IP addresses of routers on the paths to end-users. By using either DNS PTR records or augmenting `whois` information for routers with no PTR records, network domains that exist between the orchestrator and the end-users can be identified.

Step 3: Locate the IEPs: This step is central to the working of *ExEC* discovery phase. *ExEC* assumes that majority of IEPs registers their resources as a DNS service (SRV) record with a consistent service name (such as edge). We believe that this assumption is realistic as making their edge infrastructure discoverable, the IEPs can attract more clients and user traffic which can lead to increased revenue. Furthermore, IEPs have a strong incentive to ensure that the registered PTR records in DNS are accurate. An SRV record points to a server in the zone and has the following format:

```
_service._protocol.name. TTL  DNSclass SRV priority weight
     port targetname
```

Listing 1 shows an example of DNS SRV record for DNS zone `domainA` which hosts two IEPs, one MEC server – `mecServerA` and the other IEP – `edgePlatformB`.

```
_edge._tcp.domainA.com. 86400 IN SRV 10 30 5060
     mecServerA.domainA.com.
_edge._tcp.domainA.com. 86400 IN SRV 10 10 5060
     edgePlatformB.domainA.com.
mecServerA.domainA.com. 86400 IN A 192.168.121.30
edgePlatformB.domainA.com. 86400 IN A 192.168.121.31
```

Listing 1. DNS SRV Records for IEPs

The SRV records in Listing 1 advertise the entry points of IEPs supporting *ExEC*. The entry point is implemented as a RESTful service, by means of which *ExEC* IEP communicates with the orchestrator, receives containers to run, and performs other tasks.

2.3 Negotiation and Agreement

When present client flows and locations of IEPs are identified and *ExEC* orchestrator has topology at hands, it is time to negotiate about service deployments with IEPs, compute near-optimal placement of services, enter a contractual agreement with selected IEPs and finalize the deal by financial transaction.

To start negotiation procedure, the orchestrator connects to the *ExEC* management service of IEPs and sends it `hello` message, which may contain some preferences, e.g., a period for edge service deployment. In response to this message, IEPs send to the orchestrator the list of available time slots for service deployment, possible hardware configurations that edge service can utilize during those time slots, and information on a preferred way of contractual agreement. The latter one may be as simple as contact information of IEP management personnel, but we are interested in more advanced scenario enabled by smart contracts. In the case IEP prefers smart contract, it includes its address in the reply message to the orchestrator. The contract contains details of SLA, specifies a price for the deployment of service, and has other relevant information. Whether the orchestrator accepts the contract and finds offer by IEP fair, it green-lists the IEP and includes it in service placement computation, that we describe in the next section. For now, we assume that the IEP was selected for service placement. In such case, the orchestrator calls a method of smart contract that initiates a preliminary agreement with IEP and sends it positive acknowledgment message, containing information on, e.g., what slot was chosen and what is the preferred hardware. Technically, calls to such methods are posted similarly to transactions in any blockchain system and must be cryptographically signed. The execution of a method may result in the transfer of funds (cryptocurrency) from one address to another, and in our case, the result of calling the preliminary agreement method is that smart contract takes into escrow the tokens that orchestrator is supposed to pay IEP. Upon receiving positive acknowledgment IEP checks if the chosen slot is still available, and calls confirmation or rejection method of the smart contract respectively. In case everything was correct smart contract finalizes the agreement.

There is a subtle point before the funds are released from escrow and either finally transferred to the IEP or returned to the orchestrator. Namely, a malicious IEP might enter the agreement but not run the service as promised, not fulfilling the requirements of the SLA. Every open system suffers from problems alike, and *ExEC* is no exception. The typical way to treat the challenge is to have a reputation ranking for participants, as suggested in [38], that would affect the payment in the case of disagreement between IEP and the orchestrator.

2.4 Service Placement

Having topology with green-listed IEPs at hands, *ExEC* orchestrator can compute near-optimal placement of edge services. As we already mentioned, the problem of edge placement is well-studied and is mostly approached as linear programming (LP) optimization problem or an approximation of it in the case of NP-completeness. However, in our setting we choose a more practical approach, taking advantage of the fact that our topology is a tree-like structure centered at the location of the orchestrator. Namely, we suggest using betweenness centrality [17] as a metric for edge service placement. An intuition provides Figure 3, where nodes with highest

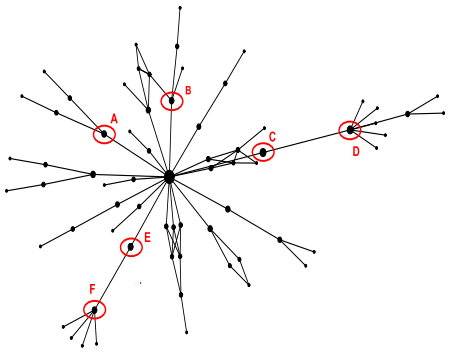


Figure 3. Network tree from cloud to clients. Nodes with highest betweenness centrality are marked red.

betweenness centrality are marked red (except the central node, where the orchestrator is located). Ideally, *ExEC* should deploy services on nodes with high betweenness centrality (i.e., the selected server serves many clients) but also which are as close to the clients as possible (i.e., minimizing latency). For example, both nodes "C" and "D" have the same betweenness centrality, but node "D" should be preferred, as it is closer to the end-users. The network graph presented in Figure 3 was constructed by sending `traceroute` probes from AWS EC2 instance running in Frankfurt (Germany) to the public IP addresses of top-100 universities from the Times Higher Education ranking [35]. We grouped the nodes into /24 subnets to simplify the graph and assumed each node to be a potential edge server location. The central node in the figure is the AWS instance, and the other points are the discovered routers or client nodes. The experiment simulates the situation where a service running on the cloud has global interest.

2.5 Redirect and Discovery of Services

In this step, the orchestrator onloads the virtualized entity containing edge service to set of available IEPs chosen for service placement. Existing clients are redirected to the closest service locations with a `302 Moved temporarily` HTTP response. HTTP `302` response allows to specify an expiration time, and this is used to ensure that the redirection is valid as long as the deployment of the service on the IEP can be guaranteed.

For new clients, it is reasonable to provide a discovery mechanism that would eliminate a round trip to the cloud. One solution would be mimicking approach used by CDNs for the discovery of content. CDNs use DNS for content discovery [37], and in fact, major cloud providers offer their own name resolution services [1, 2]. In those cases, where orchestrator and end-users are both utilizing DNS services of the cloud, it is enough that the orchestrator updates this commonly used DNS after the service migration.

2.6 Onloading Algorithm

The operation of the edge orchestrator can be implemented with three daemons as in Algorithm 1. The first one (lines 2-6) updates the topology, second (lines 7-13) discovers IEPs on the paths, computes the service placement and redeploys the services if needed, the third (lines 14-19) redirects the clients to the closest edge service deployment if available. The algorithm is presented at a high level, omitting details that parallel execution generally implies. The first daemon groups the end-users by subnets and assembles the aggregate tree by performing network tomography (e.g., utilizing

Algorithm 1 Onloading Algorithm Overview

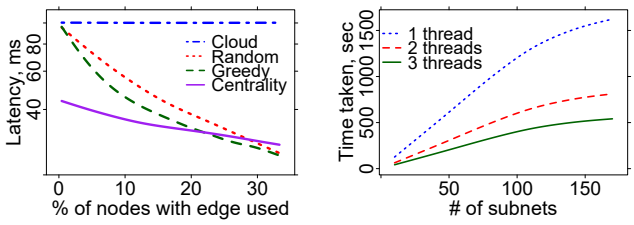
```
1:  placement ← NULL, topology ← NULL, platforms ← NULL
                                              ▷ Topology discovery daemon
2:  while TRUE do
3:      wait topology_update_interval
4:      topology ← update topology using traceroute
5:      if topology change exceeds threshold then
6:          send topology_change_signal
                          ▷ Discovery of IEPs and service placement daemon
7:  while TRUE do
8:      wait platforms_update_interval or topology_change_signal
9:      if topology ≠ NULL then
10:         platforms ← discover IEPs using DNS SRV records
11:         placement ← compute placement of services
12:         negotiate with IEPs and update placement
13:         schedule the deployment of services
                                          ▷ End-user request dispatch daemon
14: while TRUE do
15:     request ← client's request from queue
16:     if placement ≠ NULL then
17:         if placement contains edge closer than cloud then
18:             Reply with redirect to closest edge in placement
19:         else
20:             Cloud handles the request
21:     else
22:         Cloud handles the request
```

`traceroute`). If there is a dramatic change in the topology caused, e.g., by a flash crowd, the first daemon signals the second daemon. The second daemon augments the topology by discovering IEPs (line 10). Next, daemon negotiates with discovered IEPs on the conditions of service deployments and updates the placement. The third daemon dispatches the incoming clients, redirecting them to the appropriate IEP if any available. The parallel execution of three daemons enables us to perform tasks at different time granularities, depending on how quickly the conditions in the network are expected to change.

3 Preliminary Evaluation

For the evaluation, we used the public router dataset from CAIDA [21], restricted to routers located on the East Coast of the US. We ran our orchestrator from a cloud instance of Amazon Web Services (AWS), also on the East Coast. We considered routers from the CAIDA data set to be the leaves of our tree and issued `traceroute` requests to them from the cloud. Since the obtained aggregate tree was quite detailed, we grouped the nodes together by using a subnet mask of 16 bits, which resulted in a tree of 240 nodes, of which 186 were leave nodes. We considered each such node as a potential deployment point for an IEP, i.e., there were 240 possible edge server deployment points.

We first compared different placement schemes for edge services, assuming that every node has an IEP. During each round of simulation, we uniformly scattered one million clients. Using `traceroute`, we measured the average latency of the clients in four different cases of service placement: only cloud (no services placed at the edge); randomly chosen IEPs; greedily chosen IEPs (deploy service to IEP nodes with the maximum amount of clients); IEPs selected according to their betweenness centrality measure [17]. Figure 4a shows that in the absence of an edge the average latency stays around 100 ms. Centrality performs well when service can be



(a) Placement methods. **(b)** Time required by `traceroute`.

Figure 4. Performance of *ExEC*.

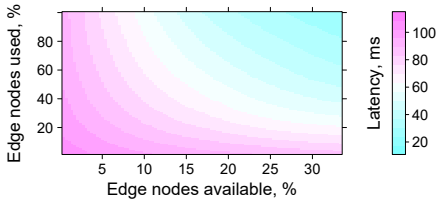


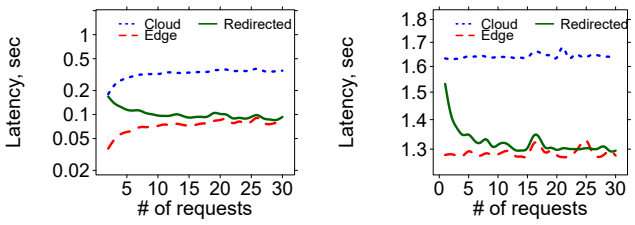
Figure 5. Performance of centrality placement given limited amount of IEPs.

deployed only to a limited number of edge nodes. Greedy method outperforms centrality when the service is deployed to over 20% edge nodes in the network. Given more than 30% of edge nodes, random placement also starts to work well. Since the application provider has to pay the IEPs, it is likely that in many cases the number of deployed services remains limited; thus betweenness centrality is an attractive choice of placement.

Next, we denounce naïve assumption that every node has an IEP and scatter limited amount of IEPs uniformly across the network. Fig. 5 shows latency for centrality placement when the underlying network has from 0 to 30% of nodes equipped with edge platforms, and application utilizes from 0 to 100% of them. We observe, e.g., that when the network has more than 20% of equipped with an edge, and around half of them with highest centrality are used by the application, the latency stays at or below 30ms.

Using a prototype client implementation, we evaluated the effect of redirection on the end-users latency. We used a laptop computer as the client, the closest Microsoft Azure facility as the cloud, and server located in the same city as the edge server. The client first sent the request to the cloud which issued an HTTP redirect to the edge server, so subsequent requests were served by the edge. We also measured latencies for using only the cloud or directly the edge server. We used two types of responses by size: a small one of 25 kB, and a large one of 1 MB. As a result for small response in Fig. 6 shows, the redirect shows good results for intense communication; when the number of requests approaches 15, the average latency becomes very close to that of the edge. In case of large response, redirect has performance gain even for a single request, and average latency of redirect converges to edge latency much faster.

Running multiple `traceroutes` takes a relatively long time, and during our experiments, we also measured the time required by `traceroute` to gather the information for building the aggregate tree. We display the results in Figure 4b. In the case of sequential execution, it took more than 1500 seconds to build the aggregate tree used in our experiments. Fortunately, `traceroute` tasks can be easily parallelized, and using three threads in parallel reduced time requirements to a third, approximately 500 seconds. Increasing the number of parallel `traceroutes` will decrease the time even



(a) Response size: 25 kB. **(b)** Response size: 1 MB.

Figure 6. Effect of redirect on end users' latency.

further. Also, since clients may reside in the same subnets, we do not have to run `traceroutes` for every single client, but can reuse the results of an earlier run. This explains the sub-linear increase in the time taken shown in Figure 4b.

4 Discussion

ExEC is not limited to computational onloading that we have just examined as the example scenario. Other use cases we discuss below are more straightforward to implement. Another commonly-used edge computing scenario is offloading computationally heavy tasks from a client to an edge server. Given the assumption of *ExEC* that IEPs are in the DNS, the client can query its current domain for the DNS SRV edge records, getting a list of nearby IEPs as a result. While the scope of such discovery is limited to the current domain, it is not a severe limitation since edge servers with minimal networking distance to the client reside nowhere but in its current domain.

The practical issue that remains open is the discovery of *context-specific* edge applications. Examples of such applications include, e.g., providing an all-around view at the stadium by aggregating individual users' video streams [27], improving the customer experience at a shopping mall [12], and many others. These kinds of applications generally make their way to the user by means of advertisement, such as billboards, emails and so on. With *ExEC*, it is possible to discover nearby IEPs, and they, in turn, can provide on client's request list of available context-specific applications and services. From a client perspective, this would require running an inquiry service, which user can turn off or on at will.

5 Related Work

The most relevant work to ours are [10, 29, 32, 36]. Bhardwaj et al. [10] propose an edge discovery protocol as a backend service that hosts a directory of available devices. However, this approach requires a centralized catalog to register the devices. Using DNS for the discovery of edge providers eliminates the problem of centralized catalog management and ownership. Varghese et al. [36] present a concept of an EaaS (Edge-as-a-Service) platform, offering a discovery protocol. The downside is that the platform relies on specific master controller nodes: to utilize edge, one must know the particular provider of EaaS to be able to discover the edge resources. *Kinaara* [32] offers discovery capabilities as an EC framework. The discovery process is enabled by special mediator nodes that keep a connection to the cloud. Yet, it is not discussed how mediators are discovered nor whether it can be used for general purpose cloud applications. Contrary to open discovery, one must know where the instance of *Kinaara* is installed to utilize its features. Mortazavi et al. suggest the concept of *path computing* [29], which is very similar

to our view of edge environment, where edge resources reside on a path from the cloud to an end user. Their framework, *CloudPath*, addresses many practical issues, but edge discovery does not happen on the fly. The developer is actually responsible for specifying the mappings between application functions and actual computing facilities in the *deployment descriptor file*. Thus, in contrast to our approach, *CloudPath* assumes static edge infrastructure which is known beforehand, already at the stage of application development. We believe that frameworks presented in [29, 32, 36] can potentially gain from utilizing *ExEC*, which removes the necessity for highly specialized components, adds more agility, and improves the overall user experience since *ExEC* discovers previously unknown resources.

MEC [30] is a standardization effort for EC in the area of mobile applications. In MEC, the edge servers are connected directly to cellular base stations making discovery of the servers easy for end-users. However, the dynamic onloading of services from the cloud is not possible since the cloud is agnostic of MEC servers. Thus, MEC could potentially benefit from *ExEC*'s discovery and negotiation protocols. In [24], the authors propose a transparent MEC deployment through middleboxes. Still, their solution relies on traffic interception, thus having narrow applicability.

Application of blockchain and smart contracts to the domain of EC is explored in [38] and [34], examining not only the realm of financial transactions but also building a decentralized control system for EC on top of the distributed ledger.

In [15] and [20], the authors emphasize the importance of building an open infrastructure for EC, concentrating on the OpenStack platform as a resource manager. In *ExEC*, IEPs can use OpenStack internally for the management of edge servers. In our recent work [40], open EC infrastructure is envisioned as a fully decentralized environment. However, we believe that retaining some degree of centralization by using the orchestrator component renders a more practical solution.

6 Conclusion

Discovery of edge platforms and dynamic reallocation of services is a vital part of making edge computing a reality. In this paper, we have presented *ExEC*, an open platform aiming to define common practices for the discovery of edge providers on the fly. With *ExEC*, we also made a first step towards establishing a standard protocol for interaction between cloud and edge providers. Our approach opens an opportunity for third-party edge providers and makes it possible for applications to unfold dynamically towards the edge without any administrative overhead. *ExEC* contributes to sustainable development by improving the utilization of existing computing facilities, enabling their usage as edge platforms.

In the future, we plan to implement the edge orchestrator providing the reference implementation of our discovery and negotiation protocols. We intend to equip the orchestrator with an online decision-making algorithm, capable of predicting user behavior and employing state-of-the-art techniques from machine learning and AI fields.

Acknowledgments

This work was supported by the Academy of Finland in the BCDC (314167), AIDA (317086), and WMD (313477) projects.

References

[1] 2019. Amazon Route 53. https://aws.amazon.com/route53/.
[2] 2019. Azure DNS. https://azure.microsoft.com/en-gb/services/dns/.
[3] 2019. Docker. https://www.docker.com/.
[4] 2019. Ethereum Blockchain App Platform. https://ethereum.org/.
[5] 2019. Kubernetes. https://kubernetes.io/.
[6] 2019. OpenStack. https://www.openstack.org/.
[7] AMD. 2019. AMD Secure Processor. https://www.amd.com/en/technologies/security
[8] ARM. 2018. ARM TrustZone. https://www.arm.com/products/silicon-ip-security
[9] Sergei Arnautov et al. 2016. SCONE: Secure Linux Containers with Intel SGX.. In *OSDI*, Vol. 16. 689–703.
[10] Ketan Bhardwaj et al. 2016. Fast, scalable and secure onloading of edge functions using airbox. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC '16)*.
[11] Flavio Bonomi et al. 2012. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 13–16.
[12] Steven Carlini. 2016. How the Internet of Things and Edge Computing Will Help Revolutionize the Shopping Experience. https://blog.schneider-electric.com/datacenter/2016/06/03/iot-edge-computing/.
[13] Alberto Ceselli et al. 2017. Mobile Edge Cloud Network Design Optimization. *IEEE/ACM Transactions on Networking* 25 (2017), 1818–1831.
[14] Lucas Chaufournier et al. 2017. Fast Transparent Virtual Machine Migration in Distributed Edge Clouds. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC '17)*. ACM, New York, NY, USA.
[15] Ronan-Alexandre Cherrueau et al. 2018. Edge Computing Resource Management System: a Critical Building Block! Initiating the debate via OpenStack. In *HotEdge 18*. USENIX Association, Boston, MA.
[16] ETSI. 2018. *MEC Deployments in 4G and Evolution Towards 5G*. Technical Report.
[17] Linton C. Freeman. 1977. A Set of Measures of Centrality Based on Betweenness. *Sociometry* 40, 1 (1977), 35–41. http://www.jstor.org/stable/3033543
[18] Pedro Garcia Lopez et al. 2015. Edge-centric computing: Vision and challenges. *ACM SIGCOMM Computer Communication Review* 45, 5 (2015), 37–42.
[19] Golem Worldwide Supercomputer. 2017. https://golem.network/
[20] David Haja et al. 2018. How to orchestrate a distributed OpenStack. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 293–298.
[21] B. Huffaker et al. 2012. *Internet Topology Data Comparison*. Technical Report. Cooperative Association for Internet Data Analysis (CAIDA).
[22] iExec. 2019. https://iex.ec/
[23] Intel. 2019. Software Guard Extensions. https://software.intel.com/en-us/sgx
[24] Chi-Yu Li et al. 2018. Mobile Edge Computing Platform Deployment in 4G LTE Networks: A Middlebox Approach. In *HotEdge 18*. USENIX Association.
[25] Tai Liu et al. 2017. The Barriers to Overthrowing Internet Feudalism. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. ACM, 72–79.
[26] Lele Ma et al. 2017. Efficient service handoff across edge servers via docker container migration. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC '17)*.
[27] Mobile Europe. 2017. Nokia brings AR to sports stadium with MEC platform. https://www.mobileeurope.co.uk/press-wire/nokia-bring-ar-to-sports-stadium-with-mec-platform.
[28] Nitinder Mohan et al. 2018. Anveshak: Placing Edge Servers In The Wild. In *Proceedings of the 2018 Workshop on Mobile Edge Communications*. ACM, 7–12.
[29] Seyed Hossein Mortazavi et al. 2017. Cloudpath: A Multi-tier Cloud Computing Framework. In *SEC '17*. ACM, New York, NY, USA.
[30] Dario Sabella et al. 2016. Mobile-edge computing architecture: The role of MEC in the Internet of Things. *IEEE Consumer Electronics Magazine* 5, 4 (2016), 84–91.
[31] Mohamed Sabt et al. 2015. Trusted execution environment: what it is, and what it is not. In *14th IEEE TrustCom*.
[32] Ahmed Salem et al. 2017. Kinaara: Distributed discovery and allocation of mobile edge resources. In *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 153–161.
[33] Mahadev Satyanarayanan et al. 2009. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing* 8, 4 (2009).
[34] Alexandru Stanciu. 2017. Blockchain based distributed control system for edge computing. In *Control Systems and Computer Science (CSCS)*. IEEE, 667–671.
[35] Times Higher Education. 2019. World University Rankings. https://www.timeshighereducation.com/world-university-rankings/2018/world-ranking.
[36] Blesson Varghese et al. 2017. Edge-as-a-Service: Towards Distributed Cloud Architectures. *arXiv preprint arXiv:1710.10090* (2017).
[37] Zheng Wang et al. 2017. Evolution and challenges of DNS-Based CDNs. *Digital Communications and Networks* (2017).
[38] Kwame-Lante Wright et al. 2018. SmartEdge: A Smart Contract for Edge Computing.
[39] Chenying Yu and Fei Huan. 2015. Live migration of docker containers through logging and replay. In *Advances in Computer Science Research, International Conference on Mechatronics and Industrial Informatics*.
[40] A. Zavodovski et al. 2018. ICON: Intelligent Container Overlays. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*. ACM, 15–21.