(to be presented at HotNets 2018)

ICON: Intelligent Container Overlays

Aleksandr Zavodovski[†] Nitinder Mohan[†] Suzan Bayhan[‡] Walter Wong[†]
Jussi Kangasharju[†]

[†]University of Helsinki, Finland [‡]TU Berlin, Germany

ABSTRACT

The Internet is largely a self-organizing system that adapts to changes in its operating environment. In this work, we extend these principles to service infrastructure and introduce ICON, standing for *intelligent container*. Technically, ICON is a container encapsulating a service that is consumed either directly by end-clients or other services. The novelty of ICON is in the ability of containers to adapt to their environment, targeting near-optimal service delivery and requiring only high-level guidance from the application management. Once deployed, containers form an overlay, observe their setting, and migrate or replicate themselves as needed, to the locations e.g. closest to service consumers. ICON captures our long-term vision for self-organizing service overlays that have the potential for global outreach. Bringing intelligence and adaptation to the level of individual containers renders a decentralized solution that has desirable properties, such as scalability, resilience, reliability, and adaptability to volatile environments. We hope that technology like ICON can open the way for more democratized service provisioning, disintermediating service providers from centralized brokers and optimizing orchestrators.

1 INTRODUCTION

The Internet is largely a self-organizing system that adapts to changes in its operating environment. While these principles have successfully been used in the networking and transport layers of the Internet, they have not yet been as widely applied into service infrastructures. New approaches and paradigms, such as edge computing [8, 32] or Service Function Chaining (SFC), all rely on a wide distribution of various services, and similar challenges are also relevant for many applications built on top of microservices architectures, running in serverless or containerized environments. Usually, the issue is addressed as an optimization problem, assuming a global view by some entity performing the optimization [10, 35]. The problem with this approach is that global information is difficult to gather or it even might not be available due to its transient nature [33].

In this paper, we introduce ICONs, short for *intelligent containers*, that are self-managing entities providing applications or services. We use the term "container" for any kind of a virtualized entity that is executed by a host. In practical terms, this could be containers, virtual machines, network functions, unikernels, etc. An ICON analyzes incoming requests and searches for the location of better (or optimal) deployment, e.g., trying to minimize latency experienced by the end-users. All ICONs for a particular service (or set of interconnected services, such as SFC) share observations of their environment and its changes via an overlay they form for the purposes of communication. Possibility to communicate enables cooperative adaptation that may pursue various (optimization) targets, such as improving the end-user experience, minimizing financial costs, or other objectives. After a better location is discovered, the container initiates a migration to this location and provides its service from there.

Given the rise of edge computing, we foresee new computation facilities, which we call Independent Edge Providers (IEPs), emerge. IEPs are independent computational facilities that provide a platform to run edge (and other) services in an open environment. IEPs can directly address new challenges, such as latency-critical edge services for AR/VR applications. IEPs can be dedicated small or midsize facilities, but even cloud providers can be considered as an extreme examples of IEPs. Multi-access edge computing (MEC) [15] servers installed by telco operators may also function as IEPs. ICONs also address the concerns expressed in [26], pointing out that crucial building blocks of the Internet are in the hands of a very few gigantic players. ICON is a technical solution that creates a market for IEPs and opens a way for a more open service provisioning on the Internet.

State-of-the-art container or VM orchestration is based on centralized controllers and is mostly limited to closed environments, whereas ICONs are intended to cross administrative borders of cluster instances and benefit from decentralized control. We believe the latter offers more flexibility, resilience, and better adaptation; we discuss these in more detail later.

In this paper, we sketch the main aspects of ICON architecture, discuss key technological requirements and show preliminary experimental results comparing ICONs with a centrally managed orchestrator.

2 ENABLING TECHNOLOGIES

ICONs depend on several key technologies and below we sketch the major requirements and solution ideas.

to be presented at HotNets-XVII, November 15–16, 2018, Redmond, WA, USA
2018. ACM ISBN XXX. . . $XX
https://doi.org/XXXX

Discovery of IEPs. Given the increase in Internet traffic [12] and the rise in edge computing, we see that offering computational facilities near the edge of the network by the IEPs is a possible trend for the future. Thus, we will need a means to enable discovery of IEPs. Here, we briefly sketch our idea of how IEPs can be located on the fly by using only existing tools and standard protocols[1]. We suggest that IEPs register themselves with their DNS servers by adding a service (SRV) record with a consistent service name (e.g., *edge*). Since their revenue comes from hosting services, IEPs have a natural incentive to add these DNS entries. Assuming that SRV records exist, ICONs can perform the network tomography to discover relevant IEPs located on the paths to the end-clients. Note that this procedure is only for discovering IEPs, and it is not intended for discovery of ICONs themselves by end-clients, which we discuss further in Section 3.3. Our discovery procedure is not intended to render an ideal map of IEPs since coarse-precision is enough in practical cases.

Migration. Capability to migrate is an essential feature of ICON. The migration of containers or virtual machines involving various scenarios (stateless, live or mobile handoff) is a well-explored topic, e.g., [11, 17, 28, 30, 48], covering all the major cases relevant for ICONs.

Security. Deployment of containers in untrusted environments makes them susceptible to security issues. While IEPs are protected by virtualization, the containers are vulnerable to the IEP examining their code and data. This is an open challenge which in some cases can be solved by Trusted Execution Environments (TEEs). Although TEEs have their limitations, Arnautov et al. [5] demonstrate how to run a Docker container securely. Migration solutions [3, 16] and microservices framework implementations [9] also exist. Some services will not require special protection, thus ICON is not conceptually dependent on TEEs.

Agreement. ICONs provide applications or services on behalf of their owners and use the computing facilities provided by IEPs. We assume that there will be a cost for the owner to have its ICONs running in an IEP. As we foresee a large market of IEPs, this will require a framework for concluding agreements and settling transactions on the fly. One candidate solution could be distributed ledger technologies (DLT) [37]. Their most attractive aspects are: i) no central authority ii) ability to make multi-party agreements using *smart contracts* [38] iii) secure, fault-tolerant, and transparent bookkeeping. A detailed description of DLT is out of this paper's scope, but from practical point of view, ICONs may utilize any platform supporting smart contracts, e.g., [42, 45]. We discuss alternatives and shortcomings of smart contracts in Section 7.

3 INTELLIGENT CONTAINER

Fig. 1 gives an overview of how ICONs operate. We refer to set of ICONs performing cooperative tasks and managed by

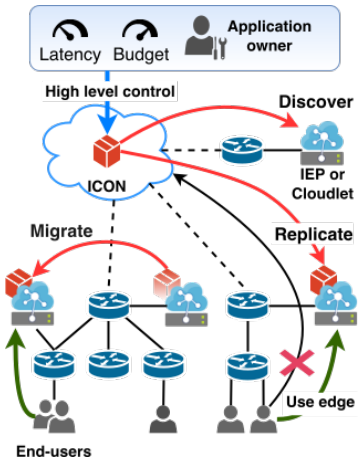

Figure 1: Overview of the system. Dashed lines are links from cloud to local subnets. Red solid arrows originating from containers show actions of ICONs. Green arrows show end-users flows switched from cloud to edge.

the same entity as an *application*. The *application owner* (or just owner) is an entity that takes strategic decisions such as adjusting performance goals according to dynamic business objectives and has full authority over the application.

We refer to the actual software that hosts ICONs as *container yard*, which can be e.g., Kubernetes-based [25], augmented with ICON-specific extensions. Fig. 1 shows ICONs starting from the cloud, however, an application may start unfolding from some cloudlet [32] or IEP. ICONs analyze incoming requests, discover possible deployment facilities (IEPs), and either migrate or replicate closer to the end-users, for example, optimizing for latency. The owner assigns a certain budget for the application and ICONs need to respect it in their decisions. There is a tradeoff between the best possible user experience and budget, and the owner can regulate this tradeoff, by controlling the budget it assigns to the ICONs. Having ICONs make independent decisions relieves the application owner from necessity to optimize a large set of containers in global deployment.

The traditional way of orchestrating such containers is to use a centralized controller (e.g., Kubernetes) in a closed environment, like a data center[2]. Extending this centralized model to a wide area network with many IEPs would be challenging as it requires essentially all of the key technologies from Section 2 for managing the containers and IEPs, and in addition, would require the application owner to specifically optimize the behavior of the containers. ICONs free the application owner from such "micromanagement" and enable the ICONs to react faster to changes in their environment.

[1]For more details, please see [49].

[2]Kubernetes has a new *federation* [24] feature, capable of joining a geo-distributed set of clusters into a unified environment, that we discuss more in Section 6.

3.1 Overlay of ICONs

To facilitate cooperative communication and control message flow, ICONs form an overlay, organized as a logical tree that we sketch in Fig. 2. The tree grows *organically* as containers deploy replicas of themselves, each replica becoming a child of its parent. To prevent orphan ICONs in the case of its parent crash, each container also has the address of the origin. Child ICONs report aggregated information to their parents: number of requests per second and other statistics can be calculated as a sum of own and descendants' requests, while latency is averaged over own and descendants' end-users. In such a way, each node can determine its own state and that of its sub-branches, while avoiding being overwhelmed by detailed data but still having enough knowledge for decision-making.

We do not explicitly treat the case of the origin ICON failing, although in such cases its immediate descendants could initiate a leader election to select an alternative. The same recovery can be used by any sibling ICONs further down the tree in case their parent crashes. However, as the origin is the main point of contact for the application, its crash is akin to a modern web application's hosting server crashing and making the application unavailable. With ICONs, we still can guarantee limited functionality where existing ICONs could continue to provide a service for (a subset of) their clients.

3.2 Control mechanisms

One of the main objectives of ICON is to make the life of application owner easier, relieving it of concerns such as discovering the best places for deployment, adjusting components individually, resorting to third parties to analyze incoming traffic, etc. However, the application owner does retain a degree of control over the ICONs it has deployed. As the ICONs form a tree, technically, it is possible for the owner to control ICONs individually, however, micromanagement is not the intended way. Instead, we propose that the application owner controls the ICONs by setting parameters, defining targets, or providing decision-making code for them. This allows each ICON to choose actions using its local information, thus making optimization problems more tractable as they would only relate to a small subset of the global network.

This flexible control system enables the potential for "intelligence" in the containers. The control mechanisms could be based on anything that the ICONs know or are able to observe, and it would allow them not only to adapt to their environment but actually learn from their actions. In this paper, we only consider simple, utility-based forms of control and adaptation, but the ICON framework is flexible enough to support more complex solutions. To evaluate the performance of an ICON, we compute the utility at a network location j which serves λ_i requests per second from a subnet i as follows:

$$\mathcal{U}_j = ((1-w)b_j + w \sum_{\forall i \in S} \lambda_i l_{i,j}) \qquad (1)$$

where b_j is the cost of running ICON at location j, $l_{i,j}$ is the latency experienced by users from subnet i if served from network location j, and w is the weight parameter ($0 \leq w \leq 1$) for tuning the importance of the cost or latency component. Function (1) is given for example purposes, and it can easily be extended by additional components for more complex scenarios. Potentially, the application owner can "hot swap" utility functions to running containers, although we assume that updating weights will be enough in most cases.

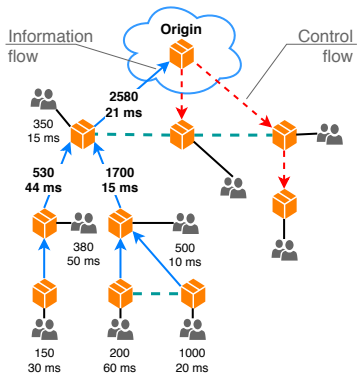

Figure 2: The ICON overlay tree. Blue solid arrows show aggregated information propagation, red dash arrows – control flow. The upper number is the number of requests per second the ICON serves, the lower number is rounded average end-user latency in milliseconds. Green dash connections between sibling are recovery links.

3.3 Discovery of the Closest ICON

A key question is how clients will discover an ICON that is closest to them. The IEP discovery mechanism sketched in Section 2 only covers the discovery of potential deployment points, not the run-time discovery of ICONs. The best method for ICON discovery depends on the environment, and below we sketch a few candidate solutions. An open issue whether we would need to find the closest ICON or if any nearby ICON would be sufficient; this is likely application-dependent.

If the network supports IPv6 anycast[3] (or some native anycast), this could be used to direct clients to their closest ICONs. The application is identified with an IPv6 anycast address, and the network takes care of locating the closest running ICON. Anycast over IPv4 is feasible, as demonstrated by the DNS and modern CDNs, but it has overhead.

A less efficient solution, but one guaranteed to work, is to use HTTP redirection. In this case, shown in Fig. 3, the new client always starts at the origin. The origin examines if any child of it is on the path of this request. If there is a descendant on the path, the ICON replies with the result of a request and redirection to its child. Sending both replies is possible in HTTP/2 [6], using server push.[4] If no child ICON is on the path, the request is served, and origin ICON may initiate IEP

[3]IPv6 supports anycast natively: packet sent to an anycast address is automatically routed to the closest node with the address [22].

[4]Without HTTP/2, the client has to go through the chain of redirects to the closest container before receiving the first byte.

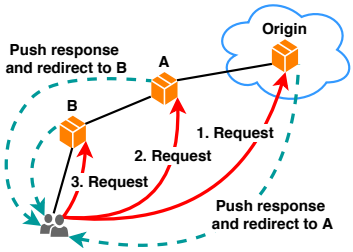

Figure 3: The redirection to the closest ICON. After the 3rd request, all subsequent requests go to the closest container – ICON B. HTTP/2 push enables sending as the response itself as well as the redirect to the next ICON.

discovery or replication to serve subsequent requests better. When a child ICON receives the next request from the client, it repeats the steps described above. As Figure 3 shows, the client will be progressively redirected to the closest ICON, although we do not immediately get to the closest ICON.

A middleground between the two solutions above would be to use DNS similarly to CDNs. This can be either mimicking the usage of DNS services by CDNs [47] or investigating third-party name resolution services, e.g. major cloud providers running their own DNS service [4, 29]. Most of these DNS and similar services are capable of dynamic updates, thus having a potential for ICON discovery.

Each of the solutions has its pros and cons, but all of them present issues when a mobile client, using a stateful service, drastically changes its point of attachment in the network. These cases would need more sophisticated discovery and migration solutions, but their overall impact would need to be evaluated in a more realistic setting. The technical details of the discovery are beyond the scope of the paper, but the above sketches can be used as a basis for a concrete solution.

4 LIFECYCLE OF AN ICON

We now describe the lifecycle of an ICON, main actions it takes during its lifetime, and rationale behind them.

4.1 Start

The lifecycle of an ICON starts with initial deployment. The application owner deploys ICON or set of ICONs to initially preselected locations, which are either cloud provider facilities or IEPs. There may be single or multiple origins, and subsequent structure will be either a tree or a forest. The application owner allocates a budget to each deployed ICON, and the ICON is free to use its budget according to the control mechanisms specified by the application owner (e.g., using the utility function the application owner determines). While we do not fix the charging model of the IEP, we assume that IEP charges some fixed amount per unit of time for having the ICON run at its location ("lease"). Once ICON has acquired a timeslot, it can run to the end of the slot at chosen location.

4.2 Active Life

While serving requests, deployed ICONs analyze incoming request patterns and discover IEPs locations along the paths to their clients (see Section 2). In this phase, ICONs actively look for better deployment locations for itself or its replicas. Depending on the actual control code, this could be trying to find a "better" place than its current place, performing optimization of placement in the subtree underneath it, or even terminating in the case of low utility.

Migration. An ICON can migrate to a more favorable location, improving its utility by e.g. minimizing end-users latency. The application owner may adjust migration behavior with *migration threshold*, specifying how much estimated utility needs to improve for migration to be a valid decision. An important aspect of migration is how the clients using the ICON at its current location will be served in the future. We leave this on the responsibility of the container yard application hosting ICONs. When migrating, the ICON will inform the yard about its new address and the yard will be responsible for redirecting clients to ICON's new location. The redirection stub can remain active for some (predefined) amount of time, depending on the properties of the application.

Replication. In the case ICON detects large request flows from a new subnet underneath it, the ICON may prefer not to leave the current location, but instead to deploy a replica of itself to handle the recently emerged user group. The new container will also be a fully-fledged ICON and will report to its parent. The application owner may adjust replication behavior in a similar way as in the previous case, using a *replication threshold*. Migration and replication are competing options, and if the ICON has enough funds, both are possible. Next, the ICON reallocates some of its budget to the replica while keeping the rest to itself; the sharing could be for example proportional to the expected utilities of the two ICONs.

Termination. If the utility of an ICON itself falls below a *termination threshold* defined by the owner, the ICON terminates itself. To preserve the logical tree structure, the descendants of terminating container will be assigned its parent. The remaining budget will be redistributed among the children of terminating ICON (or parent). Termination has the same redirection problems as migration, and we solve it similarly with redirection stub running at container yard, and remaining clients being redirected to the parent of the terminated container. The termination might be *soft*, in which the yard stores the image of the container for an agreed period of time, during which the terminated ICON may be restored promptly.

Stability. The time an ICON needs to replicate or migrate can be assumed to be known. If the changes in the environment happen faster than ICONs can react to, the system may end up in an unstable state. One solution in these cases would be for the ICONs to smoothen observations of their environments over longer time periods, to allow for stable state changes. The exact mechanisms and limits are left for further study.

4.3 Foundering

When an ICON does not have the budget to pay for its continued deployment or operation, it enters a phase of *foundering*. In other words, the ICON has run out of money. The ICON will contact its parent to ask for an increase in its budget, which the parent may or may not grant, depending on its available budget, the situation of its other children, etc. The parent may need to re-allocate the budget among its children, possibly re-shuffling the ICONs underneath it.

If the parent cannot handle the situation, it can forward the funding request up the tree, where it will eventually reach the application owner. The (human) application owner then can allocate more money for running the service or decide against it. In the latter case, there may be a re-allocation of budget across the child ICONs of the origin, and this re-allocation propagates down the tree into the sub-branches and possibly results in migration or termination of existing ICONs.

In order to avoid recurrent, massive re-shufflings, the charging model of the IEPs should be predictable, so that the ICONs can make reliable, long-term decisions about their migration and replication decisions. The idea of a fixed lease and running expense mentioned in above could be one such model, but further work would be needed to investigate various charging models and their impact on the dynamics of the ICONs under various network topologies and traffic conditions.

5 PRELIMINARY EVALUATION

In a preliminary evaluation, we compared ICON's performance with centralized orchestration in terms of communication overhead and adaptation speed. In Fig. 4 we show the number of messages required by a centralized orchestrator and an overlay of ICONs. We used the topology of European NREN provided by the Internet Topology Zoo [43]. We scattered randomly different amounts of ICONs across the graph and assumed a well-connected vertex to be the origin, which is a place where either a centralized optimizer or an origin would reside. In the centralized design, every ICON reports its load and other vital metrics directly to the optimizer. ICONs report their performance metric only to their parents. As expected, passing only aggregated data up the tree results in lower network utilization, and the growth in the number of messages for ICON is logarithmic in the number of containers, as opposed to linear for the centralized solution.

In Fig. 5, we show the results for the adaptation time of ICONs vs. a centralized optimizer. In our experiment, we assumed that the best placement of services results in 10 ms average latency for end-users. Adaptation time shows how quickly the system reaches the latency optimum after changes in user request pattern. We used the public router dataset from CAIDA [19], limited to the East Coast of the US. We derived latency data by executing traceroute from an AWS VM instance in this region, which was assumed as the location of a centralized orchestrator. As the figure shows, ICONs are able to get the required information in about 30% of the time for the centralized solution. In Fig. 5 only information

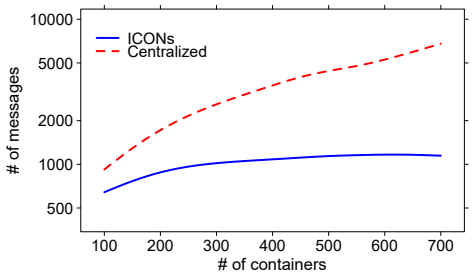

Figure 4: Number of messages.

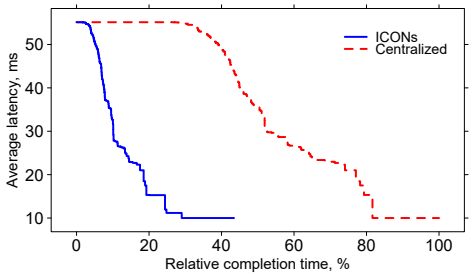

Figure 5: Adaptation time.

propagation speed is taken into account, and aspects such as the amount of data to be processed are not included. As the centralized orchestrator would need to optimize the whole tree, rather than a small sub-tree for an individual ICON, it would take more time to come up with a new deployment. Also, the time required to transfer and deploy containers is not included. For deployment, it would be similar in both cases, but the transfer of the container from cloud to edge may take longer than migration from one local IEP to another.

6 RELATED WORK

The work of Abdelwahab et al. [2] is conceptually close to ICON and applies similar ideas to IoT message queuing. In their work, IoT devices have clones to facilitate communication by queuing. Clones migrate by themselves according to current needs of the application. Our approach differs from [2] in several ways. First, we tackle a more generic problem where ICONs are general-purpose containers for running Internet services and applications. Second, we use a tree overlay instead of a swarm, minimizing messaging overhead. Third, we provide flexible high-level control mechanisms to ICON applications for changing optimization targets. We also address problems such as discovery and agreements.

The idea of cognitive IoT gateways presented in [20] is also close to ICON. IoT gateway decides whether to deploy itself to cloud or fog. The differences to our work are similar as with [2]. Running containers across multiple clouds is investigated in [1]. However, their containers lack autonomy and are governed by a centralized orchestrator.

Solutions like RightScale [31] or Turbonomic [46] aim to optimize job handling across hybrid cloud or multiple cloud providers. Compared to ICON, these are centrally governed entities relying on proprietary technologies. In contrast, we suggest an open approach without intermediary parties.

A lot of previous work in service placement has considered solving the optimal deployment of virtualized services to support QoS and SLA constraints [34]. Research has proposed semi-static placement of services in the network while addressing several metrics such as minimizing network latency [7], resource utilization [27], user-enforced policy [13, 36], etc. However, these solutions fail to consider the dynamic nature of underlying network/user traffic and rely on SDN-based approaches or even external supervision for migrating already-deployed services. On the other hand, ICONs are a better fit for requirements imposed by dynamic SFCs due to their ability to proactively react to network changes.

Platforms like Docker Swarm [14] or Kubernetes [25] can perform orchestration of containers and migrate them across nodes of a cluster. Apache Mesos [18, 40] is more versatile and supports other types of loads than containers. In our work, we suppose that IEPs will be running instances of the cluster management systems like these. Kubernetes has a new *federation* [24] feature, capable of joining a set of geo-distributed clusters into a unified environment. However, it seems unlikely that all global edge computational resources would be under control of the same cluster management system. A solution could be to have either an *orchestrator of orchestrators* or autonomous entities relying on common set of rules and conventions, able to execute in heterogeneous environments. With ICONs, we follow the latter approach, as we consider it to be more flexible. Alternatively, if some IEPs grow to become CDN-like entities provisioning all the required computational resources globally, they might still benefit from using an overlay of ICONs internally.

7 DISCUSSION

ICONs are an appealing mechanism for spreading services closer to the users and are likely to become more relevant with the advent of edge computing, where services typically reside near the users. The main power of ICONs comes from their ability to make independent decisions following the high-level guidance of the application owner.

The overlay of ICONs for a particular application can be used for collaborative sharing of environmental observations, and conflict avoidance. This information can further be used to drive any kind of more sophisticated adaptation or learning algorithms, that allow for very flexible actions of ICONs. For example, they could learn and follow the diurnal patterns and proactively place themselves accordingly. Understanding what kinds of algorithms would be suitable for this and developing solutions and algorithms are left for future work.

The control scenario we described is overly simplified, and for application consisting of interacting heterogeneous services, we would need a more complex solution. That may involve the possibility to group services by their type and have different optimization goals for each group. For example, the application may consist of latency-critical, and some back-end services, all packed as ICONs. So, the application owner may prefer to have separate "control panels" for these two groups. Another way of grouping would be to put services of various types together by their common responsibility. To illustrate, one service chain consisting of ICONs may be assigned to serve users in the U.S. and another in Europe. These two chains may have different optimization goals at different times. Large systems having specialized ICONs should have the possibility for vertically and horizontally partitioned management, while the concept of high level control inherent for ICON overlay still remains applicable and beneficial.

Another essential element in our future work is the communication between ICONs, which is especially crucial for SFC-like scenarios where the service running on one ICON needs to convey its results to a next ICON, possibly along a longer chain of ICONs. This could be implemented as part of the decision-making logic at the container, but the exact mechanics are left for further study.

While advocating the use of smart contracts for the agreements needed, we are well-aware of their shortcomings. Fortunately, problems such as slow throughput of smart contracts, high transaction costs, and energy-wasteful proof-of-work are getting addressed in [21, 23, 39, 44]. Bookkeeping transparency of distributed ledger is not always a benefit but may pose problems, e.g., for competitive reasons. In such a case, CryptoNote protocol can be used [41].

8 CONCLUSION

We firmly believe that a framework like ICON will enable new service provisioning paradigms and give a boost to edge computing, SFC and microservices architectures. ICONs make the network environment more open for entities such as IEP, that facilitate the development of cloud and edge services, weaken the monopoly of large cloud providers and, in the end, make end-user experience better. We hope that technology like ICON can disintermediate service providers from centralized brokers and optimizing orchestrators. Our next steps are: completing the design of ICON overlay, developing online decision algorithms, evaluating ICON's performance by running existing distributed applications, elaborating on security aspects. We also plan on implementing ICON using standard containers and container frameworks as building blocks and leveraging existing technologies wherever possible.

ACKNOWLEDGMENTS

This work was supported by the Academy of Finland in the BCDC (314167), AIDA (317086), and WMD (313477) projects. We wish to thank Ratul Mahajan, our shepherd, for his comments and help in improving the paper.

REFERENCES

[1] Moustafa Abdelbaky et al. 2015. Docker containers across multiple clouds and data centers. In *Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th International Conference on*. IEEE, 368–371.

[2] S. Abdelwahab et al. 2018. When Clones Flock Near the Fog. *IEEE Internet of Things Journal* 5, 3 (June 2018), 1914–1923.

[3] Fritz Alder et al. 2018. Migrating SGX Enclaves with Persistent State. *arXiv preprint arXiv:1803.11021* (2018).

[4] Amazon. 2018. Amazon Route 53. https://aws.amazon.com/route53/. (2018).

[5] Sergei Arnautov et al. 2016. SCONE: Secure Linux Containers with Intel SGX.. In *OSDI*, Vol. 16. 689–703.

[6] Mike Belshe et al. 2015. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540. (May 2015). https://rfc-editor.org/rfc/rfc7540.txt

[7] Deval Bhamare et al. 2015. Models and algorithms for centralized control planes to optimize control traffic overhead. *Computer Communications* 70 (2015), 68–78.

[8] 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.

[9] Stefan Brenner et al. 2017. Secure Cloud Micro Services Using Intel SGX. In *Distributed Applications and Interoperable Systems*, Lydia Y. Chen and Hans P. Reiser (Eds.). Springer International Publishing, Cham, 177–191.

[10] Alberto Ceselli et al. 2017. Mobile Edge Cloud Network Design Optimization. *IEEE/ACM Transactions on Networking* 25 (2017), 1818–1831.

[11] 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, Article 10, 13 pages.

[12] CISCO. 2017. Cisco Visual Networking Index: Forecast and Methodology, 2016-2021 (Whitepaper). (2017). www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf

[13] Djawida Dib et al. 2014. SLA-based profit optimization in cloud bursting PaaS. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 141–150.

[14] Docker. 2018. Docker Swarm. https://docs.docker.com/get-started/part4/. (2018).

[15] ETSI. 2018. Multi-access Edge Computing. (2018). https://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing/

[16] Jinyu Gu et al. 2017. Secure Live Migration of SGX Enclaves on Untrusted Cloud. In *Dependable Systems and Networks (DSN), 2017 47th Annual IEEE/IFIP International Conference on*. IEEE, 225–236.

[17] Kiryong Ha et al. 2017. You Can Teach Elephants to Dance: Agile VM Handoff for Edge Computing. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC '17)*. ACM, New York, NY, USA, Article 12, 14 pages.

[18] Benjamin Hindman et al. 2011. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center.. In *NSDI*, Vol. 11. 22–22.

[19] B. Huffaker et al. 2012. *Internet Topology Data Comparison*. Technical Report. Cooperative Association for Internet Data Analysis (CAIDA).

[20] Fatemeh Jalali et al. 2017. Cognitive IoT Gateways: Automatic Task Sharing and Switching Between Cloud and Edge/Fog Computing. In *Proceedings of the SIGCOMM Posters and Demos (SIGCOMM Posters and Demos '17)*. ACM, New York, NY, USA, 121–123.

[21] Joseph Poon and Vitalik Buterin. 2018. Plasma: Scalable Autonomous Smart Contracts. https://plasma.io/. (2018).

[22] Ettikan K. Karupiah and J. Itoh. 2003. *An analysis of IPv6 anycast*. Internet-Draft draft-ietf-ipngwg-ipv6-anycast-analysis-02. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-ipngwg-ipv6-anycast-analysis-02 Work in Progress.

[23] Kieran Smith. 2018. Ethereum's move to PoS - First version of Casper released. https://bravenewcoin.com/news/ethereums-move-to-pos-first-version-of-casper-released/. (2018).

[24] Kubernetes. 2018. Federation. https://kubernetes.io/docs/concepts/cluster-administration/federation/. (2018).

[25] Kubernetes. 2018. Production-Grade Container Orchestration. https://kubernetes.io/. (2018).

[26] 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.

[27] Marcelo Caggiani Luizelli et al. 2015. Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 98–106.

[28] Lele Ma et al. 2017. Efficient service handoff across edge servers via docker container migration. In *SEC*.

[29] Microsoft. 2018. Azure DNS. https://azure.microsoft.com/en-gb/services/dns/. (2018).

[30] OpenVZ Team at Virtuozzo. 2018. Checkpoint/Restore In Userspace (CRIU). https://criu.org/Docker. (2018).

[31] RightScale. 2018. RightScale Cloud Management Platform. https://www.rightscale.com/products-and-services/products/cloud-management-platform. (2018).

[32] Mahadev Satyanarayanan et al. 2009. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing* 8, 4 (2009).

[33] Michael Schapira and Keith Winstein. 2017. Congestion-Control Throwdown. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks (HotNets-XVI)*. ACM, New York, NY, USA, 122–128.

[34] Mohamed Abu Sharkh et al. 2013. Resource allocation in a network-based cloud computing environment: design challenges. *IEEE Communications Magazine* 51, 11 (2013), 46–52.

[35] P. Silva, C. Perez, and F. Desprez. 2016. Efficient Heuristics for Placing Large-Scale Distributed Applications on Multiple Clouds. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*.

[36] A. Silvestro et al. 2018. MUTE: MUlti-Tier Edge Networks. In *Proceedings of the 5th Workshop on CrossCloud Infrastructures & Platforms (CrossCloud'18)*. ACM, New York, NY, USA, Article 1, 6 pages.

[37] Melanie Swan. 2015. *Blockchain: Blueprint for a New Economy*. O'Reilly Media, Inc.

[38] Nick Szabo. 1996. Smart Contracts: Building Blocks for Digital Markets. http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html. (1996).

[39] Jason Teutsch et al. 2018. TrueBit: A Scalable Verification Solution for Blockchains. https://truebit.io/. (2018).

[40] The Apache Software Foundation. 2018. Apache Mesos. http://mesos.apache.org/. (2018).

[41] The CryptoNote Foundation. 2018. CryptoNote. https://cryptonote.org/. (2018).

[42] The Ethereum Foundation. 2018. Ethereum: Blockchain App Platform. https://ethereum.org/. (2018).

[43] The Internet Topology Zoo. 2018. European NREN Model. http://www.topology-zoo.org/eu_nren.html. (2018).

[44] The IOTA Foundation. 2018. IOTA: The Next Generation of Distributed Ledger Technology. https://www.iota.org/. (2018).

[45] The Linux Foundation. 2018. Hyperledger. https://www.hyperledger.org/. (2018).

[46] Turbonomic. 2018. Turbonomic Workload Automation for Hybrid Cloud. https://turbonomic.com. (2018).

[47] Zheng Wang et al. 2017. Evolution and challenges of DNS-Based CDNs. *Digital Communications and Networks* (2017). http://www.sciencedirect.com/science/article/pii/S2352864817300731

[48] 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*.

[49] Aleksandr Zavodovski et al. 2018. eDisco: Discovering Edge Nodes Along the Path. *arXiv preprint arXiv:1805.01725* (2018).