Title

MECANO: Integrated Measurement of Compute and Network Operations

Authors

Albalawi, A.
Westphal, C.
Chakraborti, A.
et al.

Publication Date

2019

Peer reviewed

MECANO: Integrated Measurement of Compute and Network Operations

Abdulazaz Albalawi*, Cedric Westphal*†, Asit Chakraborti†,
Dirk Kutscher‡, Jeffrey He§, J.J. Garcia-Luna-Aceves *
*Department of Computer Engineering, University of California, Santa Cruz, USA
†Futurewei Technologies, Santa Clara, USA
‡Emden University, Emden, Germany
§Huawei, Shenzhen, China
aalbalaw@ucsc.edu, cedric@soe.ucsc.edu, asit.chakraborti@futurewei.com,
dk@dkutscher.net, jeffrey.he@huawei.com, jj@soe.ucsc.edu

***Abstract*—In-network computing is a growing area of research where computations are more tightly integrated with the networking substrate and placed dynamically within the network at appropriate locations. However, this yield new issues in the domain of measurements and networking: it adds a dimension in the measurement domain, namely that of the computation for some tasks. We discuss MECANO, an architecture that integrates application measurement with network measurement to optimize the placement of workloads in the network. We have implemented an initial proof of concept of this architecture and validated its performance in our testbed. In particular, keeping track at the network layer of the location of running instances of a specific function can significantly speed up the completion time for a specific client request.**

I. Introduction

Computation is being integrated with the networking substrate [20]. The traditional model has evolved from computation happening on a single server, to the data center, and now to being distributed at the edge. An application is no longer running on a stand-alone dedicated server, but in a virtual machine or a container that can be placed on any eligible node, and moved based upon some of the application requirements.

In particular, virtualization allows software to decouple the application from a dedicated server, and the joint optimization of the networking and workload placement is now feasible. This trend has been embraced by operators with the advent of Network Function Virtualization (NFV), which enables locating network functions where needed, and scaling up and down the amount of resources allocated to a specific application.

This shift requires a similar shift in the way network monitoring and measurement are implemented. Indeed, an approach that focuses exclusively on the network would yield valuable information, but would view only be the tip of the iceberg.

We contend that, as a consequence of the integration of networking and computing, network measurements should extend into the application layer as well. The performance of the network depends on the joint optimization of the function placement with the underlying network fabric, and similarly the measurement infrastructure should align itself with this joint optimization.

The key contribution of this paper is to discuss the joint monitoring of network and application functions, as this convergence is required by the tighter integration of the two layers. We show how some basic measurements at the application layer can be leveraged to optimize the service of clients' requests in a manner that satisfies both the application requirements and the optimization of the network resources.

We present MECANO (for MEasurement of Compute And Network Operations), a monitoring layer that measures both network performance and application-layer events. MECANO is a first step in the direction of integrated measurements of networking and computing. MECANO is implemented on top of the Sapphire [28] architecture, so as to demonstrate the feasibility of our proposal.

The rest of this paper is organized as follows: Section II motivates the need for our architecture and discusses related work. Section III describes the design of MECANO, and Section IV presents the details of its implementation. Section V evaluates the performance of MECANO to show the benefit of joint network and application monitoring using a testbed. Finally, Section VI concludes the paper.

II. Motivation and Related Work

A. Related Work

The performance of a service that can be instantiated on demand at multiple distributed locations clearly depends on the underlying network. However, it also depends on the availability of this service at the chosen location, in the sense that: (a) the actual code for the VM/container/microservice may have to be fetched to run at this specific place, and (b) the service can be in different states of readiness as in most high availability systems. If the service is in a cold state at a node, then although the binaries for the service exist at the node, the service has to be installed and configured to run. If the service is in a warm state at a node, then the service is installed but not running. Lastly, if the service is running and available at a node, it is said to be in a hot state.

The proper allocation of the workload to compute servers distributed over a network is required to maximize the performance of the network. Clients are typically placed at the edge and may require applications that are delay-sensitive and require an optimized placement of the function to perform properly. If the application imposes stringent latency requirements, then the mechanism assigning the workload to a location should be aware of both the network performance and the availability of the function.

There has been considerable effort to integrate networking and computing in the research literature. Networking research [22] has considered some of the aspects to enable this joint integration.

Programmability in the network is being supported by such protocols as P4 [7]. The P4 framework provides programmability to switches in the network, but is restricted to computations that are achievable via a match-action switch architecture. This builds upon the work by Song [23], which introduced protocol-oblivious forwarding (PoF). One important use case for P4 is that of network telemetry, as it pertains to measurements.

Wang et al. [25] provide a survey of mobile edge networking research. They classify edge networks into roughly four categories and consider the issues of computation offloading and caching in each categiry. The four categories they consider are mobile cloud computing [15], mobile edge computing [5], [8], [12], fog computing [6], [10], and cloudlet [21]).

Roberts [19] gives a comprehensive overview of serverless computing, whch instantiates function on demand, albeit primarily stateless functions. He describes serverless-computing working mechanisms, advantages, drawbacks, and future directions.

Freedman et al. [9] propose an architecture that supports flow-based anycasting to services and allows for these services to be mobile. The work on SERVAL [17] similarly exposes a new Service Access Layer (SAL) that sits above an unmodified network layer. This enables applications to communicate directly on service names. Service Oriented Networking [11] provides the mechanisms required to deploy a replicated service instance in the network and to route client requests to the closest instance in an efficient manner.

Tschudin and Sifalakis [24] introduce Named-Function Networking (NFN), an extension of Named-Data Newtorking [14] that supports invocation of function by name. NFN uses the lambda-calculus syntax to express functions, and their input parameters. The lambda-calculus framework provides the required expressivity to include a wide range of potential functions. Krol et al. [16] use unikernels to deploy Named Functions as a Service.

Delay-sensitive applications include video [27] or AR/VR, for which MECANO could be a tool for better network support [13], [26].

MECANO is related to this previous work. However, none of the prior approaches we have mentioned provides a comprehensive framework to support the joint optimization of networking and computing, and in particular to explicitly make function awareness a measurement objective.

B. Architectural Considerations

1) Adding Function Awareness:

To support the use case for an integrated in-network computing, we need an integrated measurement plane as well. We suggest the introduction of monitoring modules that interact with the network layer to provide some function awareness.

The function monitoring can be very simple or more complex. In the most basic sense, the network could be aware that a specific function can be executed at a specific location. For instance, the monitoring would be aware that a function is located at a given host, or that is has migrated or been duplicated to another host. This is a simple binary answer to the question of the availability of a function at a specific node.

With the proper hooks from the server onto the network, this could be enhanced with some characterization of this availability, for instance that the function is active, or in warm stand-by, or only that the ”cold” code is there to be loaded if needed.

In the most advanced sense, the network would be able to monitor the network performance combined with, say, the completion time of the function, providing a combined measure of the performance.

Here, we focus on the basic approach as a motivating example for a joint network and function measurement layer.

2) In-network, or application layer measurement and monitoring?:

Mechanisms already exist to deploy microservices on demand in the data center. Amazon Lambda [1] is an example of instantiating some services on demand. Service meshes [4] such as Istio [3] or Envoy [2] provide some of the features of such network-compute integration, but are generally agnostic of the underlying network. The network they consider is the data center where they are located. The network performance is typically predictable, and the measurement plane can focus on application provisioning.

At the other end of the spectrum, network protocols have been enhanced with service awareness to support service-centric networking. SCAFFOLD [9] proposed an architecture that supports flow-based anycast to services, and allowed these services to be mobile. SERVAL [17] similarly exposes a new service sccess layer (SAL) that sits above an unmodified network layer. This enables applications to communicate directly on service names. Named Function as a Service (NFaaS) [16] is a framework that extends the Named Data Networking architecture to support in-network function execution. Similarly, Named-Function Networking (NFN) [24] is an extension of Named-Data Newtorking that supports invocation of function by name. These approaches add service awareness onto a network stack.

The latter approaches would require to bring some application awareness onto the network, for routing purposes in an immediate step, but also for measurement and monitoring so that the network can deliver on its objectives.

III. MECANO

We propose MECANO to provide a joint network and function measurement mechanism; namely MECANO needs to provide *network and function awareness* in order to deliver a joint optimization of the network and compute layers.

A. Overview

MECANO is designed as an instrumentation mechanism to expose to the function allocation mechanism a view of the network that includes some function information. In our first iteration of MECANO, we provide only an availability indication. This can be enhanced with more features in subsequent iterations.

Figure 1 shows the functional components of the MECANO framework. The next subsections describe the functional elements in more detail, namely the controller, the monitoring agent and the node architecture.

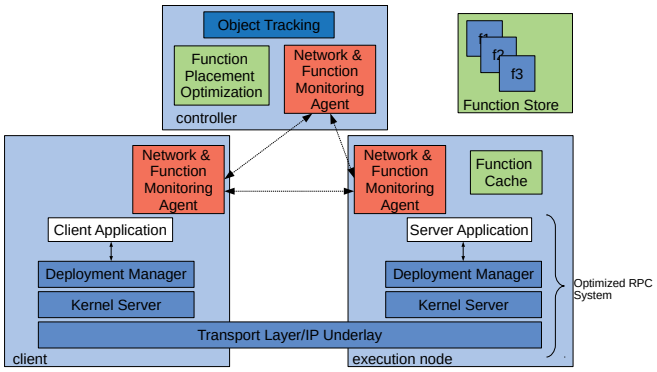

Fig. 1: Functional elements of MECANO

B. Controller

MECANO relies on the existence of a distributed controller that allocates resources for a client request. Here we assume a hierarchical controller architecture: Edge controllers manage the resource and the objects located at a specific edge cluster. These edge controllers may register with one or more cloud controllers. A lightweight protocol synchronizes the edge controller with the cloud controller(s), and specifies what resources are available in the cloud to the edge controller. This is similar to the control plane that any monitoring infrastructure would have to update.

Clients register with the controller located at the edge node. Edge controller discovery can be achieved using a variety of mechanisms that we do not discuss here.

To keep allocation of resources manageable, we constrain it to the edge controller being able to forward service requests to the cloud controller. This allows the system to reduce the amount of information to transmit between edge and cloud controllers.

The edge controller performs the authentication of the clients (through any available mechanism) and ensures that the functions requested by the client are consistent with the access control and other applicable policies.

The controller tracks the location of the workloads. The edge controllers track the location and properties of all the workloads that were issued by one of its clients; the cloud controllers track the workloads that their own clients requested and the objects instantiated by their dependent edge controllers within the cloud cluster.

C. Monitoring Agent

The controller is enhanced with a network and function monitoring agent. This is the core of the measurement infrastructure. This agent probes the network between edge controllers that are adjacent, and between the edge controllers and the cloud controllers that they are dependent of. This is performed via a combination of active probing and passive monitoring of messages sent among these controllers. The probing period is a parameter. In our implementation, we vary this parameter and find that 3 seconds is a good trade-off between frequency and accuracy.

The monitoring agent listens for requests for specific functions, and keeps track of which function is being run at the execution node. It can then report to the controller (edge or cloud) what functions are available at the node.

D. Node Architecture

Each node in MECANO support a kernel server (KS) that includes a network and function monitoring agent. This is true not only for the controller, but for the client edge, the execution nodes and some forwarding nodes in between. This agent periodically relays the network conditions to the controller that it is associated with, as well as the availability of functions (where each function is associated with a unique ID).

IV. Implementation

Sapphire [28] is a distributed programming platform that makes it easy to program mobile and cloud applications. Sapphires key design feature is its distributed runtime system, which supports a flexible and extensible deployment layer for solving complex distributed systems tasks, such as fault-tolerance, code-offloading, and caching. Sapphire is an open source project and is available on GitHub. However, Sapphire has no network awareness, and the controller only keeps tracks of actual workloads, not of the location of the executables/VM/containers. We implemented a proof of concept of MECANO as an extension of the Sapphire code base. We have built network monitoring agents that report network conditions as well a function availability.

We also implemented a few specific functions to test our approach, including a face recognition application and some HTTP server applications in addition to leveraging existing applications built for Sapphire. Sapphire is designed so that application developers need to specify the application logic and the deployment logic in a separate but structured way. By providing deployment abstractions, Sapphire hides from the application developer the mechanisms to distribute the components of the application. This is attractive, as we can

leverage these mechanisms to integrate the placement of the application components within the network.

In particular, Sapphire allows application developers to wrap their function into Sapphire Objects (SO). These SOs are then associated with a deployment manager that specifies how the function can be distributed in the network. Sapphire provides a library of deployment managers that have different characteristics to support application-specific deployment needs. Deployment managers can specify where to deploy the SOs, what type of reliability to provide, where to cache the data, what consistency level is required, and other deployment aspects.

Functions are implemented in JVM or in containers using the GraalVM framework. This allows for ease of application development and for code reuse over multiple platforms.

Upon receiving a request for a service, the edge controller performs an optimization step to decide where to allocate the service. It makes this optimization based upon the knowledge of the availability of function provided by the MECANO monitoring infrastructure.

Using the network-condition information and the awareness of the function location allows the controller to allocate the service to the appropriate kernel server.

Periodic monitoring of the kernel servers may trigger a service migration to improve performance, based upon network condition, and the potential availability of a better suited execution node for the workload.

MECANO tracks what functions are available locally and in adjacent clusters. Functions can be in one of several states: non-existent (which means the function is not available locally and would need to be downloaded and instantiated), cold (as in binary code that is stored but not loaded), or warm (for the active functions). The controller uses this information for delay sensitive application, potentially forwarding requests to warm instances rather than cold or non-existent.

Each supported node in Sapphire must provide a *deployment kernel* that tracks, addresses, and potentially migrates the SOs. Calls from one SO to another are routed by the deployment kernel.

Sapphire also provides a centralized coordinator, denoted here as the Object Management System (OMS). An object that requires global coordination registers with the OMS, and hence Sapphire knows where all objects are located and how to connect them. Without proper care, such coordinator could become a scalability bottleneck, as we will discuss below.

We created network-monitoring agents and function-monitoring agents. The OMS is periodically updated by these monitoring functions about the network performance and the availability of the functions. When a client requests a function, the OMS can therefore make an allocation decision that is a joint optimization.

V. Evaluation

We evaluated MECANO under different scenarios to validate the benefit of the proposed approach using our Sapphire-based implementation.

We consider a basic topology that consists of two edge clusters, and a cloud network. The clients connect to the network via the edge clusters.

Our evaluation scenario (see Fig. 2) demonstrates the benefit of having the ability to optimize the workload placement based upon the availability of the specific function at a node. In this scenario, several clients using latency sensitive applications connect to an access network, which in turn connects to a set of edge kernel servers and a set of cloud kernel servers. A latency-sensitive application here means that the client specifies in its request that the controller should return the result as fast as possible.

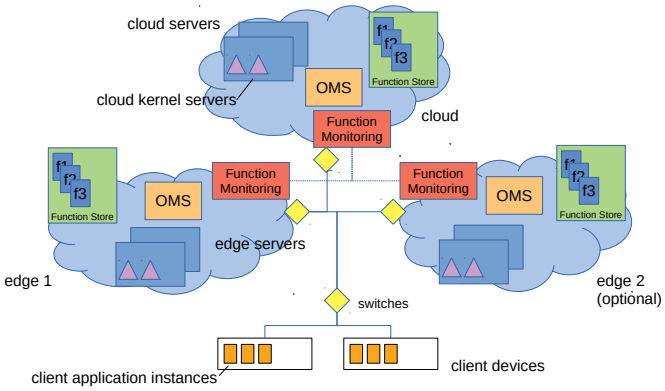

Fig. 2: MECANO Evaluation Testbed

The model of this workload is based on the HTTP distribution of requests indicated by Pries and Tran-Gia [18] with a slight modification to increase network load for our evaluation. Initially, none of the kernel servers have the server code locally. The code modules are retrieved from a function repository (HTTP) server in the cloud. The topology we consider consists of one edge with ten kernel servers, and an HTTP server located at the cloud. To simulate a real network scenario in our evaluation, we used fixed latencies between clients, edge, and cloud, where latency between clients and cloud is higher than client to edge.

Fig. 3: Performance with awareness of code location

Without MECANO, the tasks are allocated with a round-robin policy. With MECANO, the tasks are assigned preferentially to the kernel servers that have executed the given application in the recent past. This is done by keeping track of which functions were deployed by each server. For this scenario, we considered ten clients requesting a latency-sensitive application. Figure 3 shows the service-access latency, defined as the time that elapses before a service is responsive after a client request for service creation is made. As the figure shows, MECANO outperformed the vanilla cases for all ten clients, becaue MECANO tries to reuse kernel servers that are "warm," namely with the code downloaded and ready to execute.

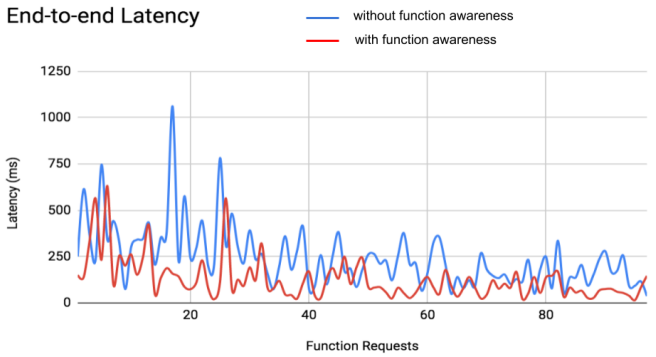

Fig. 4: Function Awareness: Latency

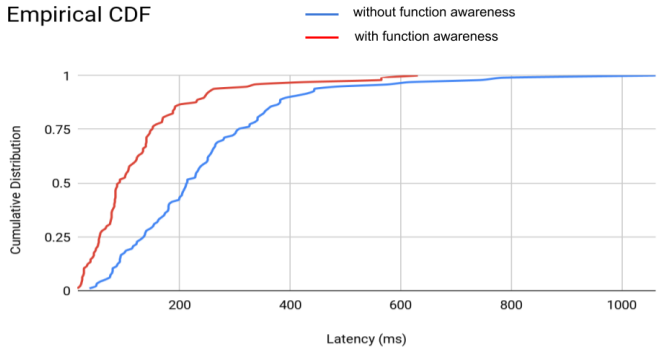

Fig. 5: Function Awareness: CDF

Next, we considered 100 clients requesting to deploy a latency-sensitive application. In this evaluation, the controller takes into account the observed network latency from similar clients to similar servers and the availability of the requested function. As we mentioned earlier, the monitoring agent in MECANO keeps track of the kernel servers that have executed the given application in the recent past, and tasks are primarily allocated to these servers. Also, they also measure the service access latency (as defined above) and report this to the controller to help optimize function placement in the network. As seen in Figure 4, MECANO outperforms the case where function placement and service access latency were not taken into consideration. This is because MECANO tries to reuse kernel servers that are "warm," namely with the code downloaded and ready to execute. This can also be seen in Figure 5, which shows the empirical CDF. As shown, 75% of the function requests for all clients were satisfied in less than 150ms, whereas in the other case only 30% of the client's requests were satisfied within 150ms.

Our evaluations show a major benefit in just knowing where functions have been deployed, and feeding that information back to a controller. However, the granularity of the information shared with the controller could be much finer: application statistics could be measured, such as the distribution of the completion times in a given context.

VI. Conclusions

We have presented MECANO, a framework for MEasurement of Compute And Network Operations. We have argued that the measurement operations in the network must become more and more integrated with the applications, in particular since containerization, virtualization and microservices allow services to be quickly migrated and jointly optimized with the underlying network. We have presented an architecture for monitoring the network performance as well as the availability of specific functions at given execution nodes, using distributed network- and function-monitoring agents.

We have implemented parts of MECANO leveraging the features of Sapphire [28], an existing distributed programming framework. Our evaluation results on an actual system clearly show the major benefits of a joint network-and-compute optimization. In particular, or results show that using knowledge of which functions are in hot standby results in a significant reduction in the time needed to complete executing the applications.

References

[1] Aws lambda - resources. https://aws.amazon.com/lambda/resources/. (Accessed on 09/23/2018).

[2] Envoy. https://www.envoyproxy.io/. (Accessed on 01/30/2019).

[3] Istio. https://istio.io/. (Accessed on 09/23/2018).

[4] Service mesh and microservices networking. https://www.datocms-assets.com/2885/1536681707-consulwhitepaperaug2018.pdf. (Accessed on 09/23/2018).

[5] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5:450–465, 2018.

[6] A. Ahmed and G. Pierre. Docker container deployment in fog computing infrastructures. In *IEEE International Conference on Edge Computing*, July 2018.

[7] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[8] M. P. et al. Mobile-edge computing - introductory technical white paper. In *European Telecommunications Standards Institute (ETSI)*, September 2014.

[9] M. Freedman, M. Arye, P. Gopalan, S. Y. Ko, E. Nordström, J. Rexford, and D. Shue. Service-centric networking with scaffold. In *Princeton University, Computer Science, Technical Report TR-885-10*, 2010.

[10] N. K. Giang, R. Lea, M. Blackstock, and V. C. M. Leung. Fog at the edge : Experiences building an edge computing platform. In *IEEE EDGE*, July 2018.

[11] D. Griffin, M. Rio, P. Simoens, P. Smet, F. Vandeputte, L. Vermoesen, D. Bursztynowski, and F. Schamel. Service oriented networking. In *2014 European Conference on Networks and Communications (EuCNC)*, June 2014.

[12] I. Hadžić, Y. Abe, and H. C. Woithe. Edge computing in the epc: A reality check. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, SEC '17, pages 13:1–13:10, 2017.

[13] D. He, C. Westphal, and J. Garcia-Luna-Aceves. Network support for ar/vr and immersive video application: A survey. In *ICETE SIGMAP*, July 2018.

[14] 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*, CoNEXT '09, 2009.

[15] A. U. R. Khan, M. Othman, S. A. Madani, and S. U. Khan. A survey of mobile cloud computing application models. *IEEE Communications Surveys Tutorials*, 16(1):393–413, 2014.

[16] M. Król and I. Psaras. Nfaas: Named function as a service. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*, ICN '17, 2017.

[17] E. Nordström, D. Shue, P. Gopalan, R. Kiefer, M. Arye, S. Ko, J. Rexford, and M. J. Freedman. Serval: An end-host stack for service-centric networking. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, San Jose, CA, 2012. USENIX.

[18] Z. M. R. Pries and P. Tran-Gia. An HTTP web traffic model based on the top one million visited web pages. Next Generation Internet, pages 133–139, 2012.

[19] M. Roberts. Serverless architectures. *online: https://martinfowler.com/articles/serverless. html*, 4, 2016.

[20] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis. In-network computation is a dumb idea whose time has come. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets-XVI, pages 150–156, 2017.

[21] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, Oct. 2009.

[22] H. Schulzrinne. Networking research - a reflection in the middle years. *Computer Communications*, 2018.

[23] H. Song. Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 127–132, 2013.

[24] C. Tschudin and M. Sifalakis. Named functions and cached computations. In *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, Jan 2014.

[25] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang. A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access*, 5, 2017.

[26] C. Westphal. Challenges in networking to support augmented reality and virtual reality. In *IEEE ICNC*, Jan. 2017.

[27] Z. Yan, M. Zhao, C. Westphal, and C. W. Chen. Towards guaranteed video experience: Service-aware downlink resource allocation in mobile edge networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 2018.

[28] I. Zhang, A. Szekeres, D. Van Aken, I. Ackerman, S. D. Gribble, A. Krishnamurthy, and H. M. Levy. Customizable and extensible deployment for mobile/cloud applications. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, 2014.