Hyperprofile-based Computation Offloading for Mobile Edge Networks

Andrew Crutcher*, Caleb Koch†, Kyle Coleman‡, Jon Patman§, Flavio Esposito‡, Prasad Calyam§
*Southeast Missouri State University, alcrutcher1s@semo.edu
†Cornell University, cak247@cornell.edu
‡Saint Louis University, {kylecoleman, espositof}@slu.edu
§University of Missouri, jpxrc@mail.missouri.edu; calyamp@missouri.edu

***Abstract*—In recent studies, researchers have developed various computation offloading frameworks for bringing cloud services closer to the user via edge networks. Specifically, an edge device needs to offload computationally intensive tasks because of energy and processing constraints. These constraints present the challenge of identifying which edge nodes should receive tasks to reduce overall resource consumption. We propose a unique solution to this problem which incorporates elements from Knowledge-Defined Networking (KDN) to make intelligent predictions about offloading costs based on historical data. Each server instance can be represented in a multidimensional feature space where each dimension corresponds to a predicted metric. We compute features for a "hyperprofile" and position nodes based on the predicted costs of offloading a particular task. We then perform a k-Nearest Neighbor (kNN) query within the hyperprofile to select nodes for offloading computation. This paper formalizes our hyperprofile-based solution and explores the viability of using machine learning (ML) techniques to predict metrics useful for computation offloading. We also investigate the effects of using different distance metrics for the queries. Our results show various network metrics can be modeled accurately with regression, and there are circumstances where kNN queries using Euclidean distance as opposed to rectilinear distance is more favorable.**

***Keywords*—Knowledge-defined networking, machine learning, computation offloading, mobile edge networks, k-Nearest Neighbor**

I. Introduction

Internet of Things (IoT) technologies introduce the need for energy-aware and latency-sensitive management strategies to ensure reliable performance in resource constrained environments. One such situation is disaster incidents where first responders may be operating in areas with limited network and computing resources. Disaster response teams may also benefit from having sensor and visual data processed on site by utilizing computation offloading strategies to make optimal decisions based on energy or latency requirements of the user.

The computing environment of disaster response networks is similar to general edge computing networks in that we have pervasive computing infrastructure with multi-modal, multi-dimensional, and geospatially dispersed data sources that rely on a wide range of services (e.g. pedestrian tracking, facial recognition, location services) [1]. Typically, the main challenge of edge computing is concerned with how to execute these services on resource constrained devices such as mobile phones or other IoT devices.

A popular and well-studied resolution to this challenge is computation offloading where resource intensive tasks are migrated to nearby cloud or edge servers with abundant computing resources. This is necessary because mobile devices are limited in terms of battery life, wireless communication, and computing capabilities [2]. Computation offloading is ideal because it typically results in lower energy consumption and processing time for the mobile user [3]. Broadly speaking, computation offloading can offset the limitations of resource constrained mobile devices thereby offering a greater variety of services to the user [4].

The control mechanism for computation offloading has been a popular research topic and several offloading frameworks have been proposed [5] [6]. There also exists a desire to develop intelligent, runtime offloading schemes to make decisions regarding when and how to offload [7]. A new emerging paradigm known as Knowledge-Defined Networking (KDN) relies on Network Analytics (NA) and Software-Defined Networking (SDN) to efficiently learn stateful information about a network [8]. KDN makes use of NA to build a high-level model of the system known as the *knowledge plane* [9].

Given the heterogeneous nature of edge networks, we can employ machine learning (ML) techniques to understand relationships between relevant variables that other analytical approaches may fail to capture. However, ML is only feasible if accurate training data is available. Traditionally, this is an issue as networks are inherently distributed systems and nodes have limited view and control of the network. However due to the development of SDN, the control and data planes can be decoupled which allows for a logically centralized control plane. SDN offers not only control of the network but also the ability to collect training data for the ML model.

In this paper, we aim to study the benefits of using KDN concepts to guide the design of an intelligent computation offloading framework. By intelligent we imply that our framework uses historical data to build a predictive model that can encode system and user dynamics that other deterministic heuristics may fail to capture. We design various network simulations in ns-3 in order to create a robust dataset that is used to train an ML model. We account for mobility by varying distances between the user and access points. The predictions from the model can then be used as input features in a multi-dimensional space we call the *hyperprofile*.

The hyperprofile consists of a set of nodes that correspond to physical machines which are positioned based on the predicted performance of that server for a given task. The user is represented in the feature space such that a query on the user's representation returns a set of nodes to which we offload the task. Our results indicate that the query method can play

an integral role in scheduling tasks to offload to nodes. We provide a mathematical basis for why the Euclidean distance metric tends to favor nodes with a balanced trade-off between features.

The rest of the paper is organized as follows: Section II discusses previous work on computation offloading frameworks. In Section III we formalize the offloading problem, Section IV details our KDN-based model for selecting optimal nodes for offloading. Different query methods are discussed in Section V, and Section VI concludes the paper.

II. Related Work

A. Computation Offloading in Mobile Edge Networks

Mobile Edge Computing (MEC) can provide an energy-aware solution to computation offloading for IoT devices where energy conservation is more desirable than low-latency [10]. The mobile users could take photographs of victim's faces and then perform facial recognition to identify the victim. The computation involved for facial recognition happens on either a core cloud or an edge computer near the user. However, many offloading frameworks only account for having one edge computer which limits the user in terms of mobility [10] [5]. They also rely on the user to decide what offloading strategy would be best for their situation. We posit that such decisions are most optimally handled by an ML model rather than someone unfamiliar with the structure and operation of networks (e.g. a first responder).

There are a number of existing offloading strategies. Yang *et al*. studied the problem under multiple mobile device users sharing a common wireless connection to the cloud. Their solution uses a genetic heuristic algorithm and focuses on *code partitioning* and deciding whether to offload each partition individually. Researchers in [3] formulate the offloading problem as a multiple choice knapsack problem where one is trying to maximize bandwidth utilization subject to constraints such as battery life. An optimization approach is also taken in [11], and their solution utilizes Lagrange multipliers.

It is difficult to adapt these solutions in real-time when some metrics may be unavailable or when the user needs change. A key feature of our solution is that it can be adapted based on the metrics available. This adaptability is achieved by effectively *decoupling* the process of developing a network model (discussed in Section IV) and incorporating the user into that model to suit his/her needs best (discussed in Section V).

B. Knowledge–Defined Networking

Mestres *et al*. note in [8] that ML models could work well with managing network behavior if the training data adequately represents the network itself. However, the authors remark it is unclear what constitutes representative training data in networking. This is left as an open research problem. On a fundamental level, our paper is motivated by the question of *what characteristics of a network are relevant for developing ML models?* We believe our approach to this problem is unique because (1) we focus on the performance of *feature selection* to develop a hyperprofile space and (2) then we apply kNN to find optimal destination nodes for offloading. We describe each formulation in depth in future sections (namely sections IV-B and IV-D).

III. Problem Formulation

Coordinating how computation is offloaded to edge servers can be seen as a job shop scheduling problem which is a popular problem in computer science literature [12]. Job shop scheduling is concerned with optimally scheduling a set of jobs on machines with varying processing constraints. It minimizes some objective such as energy consumption or makespan (total time to process all jobs) [13].

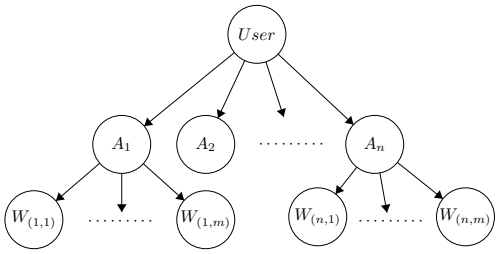

Fig. 1: Job scheduling problem represented as a multi-tier partition/aggregate application structure.

We can further extend the concept of relating computation offloading to a job shop scheduling problem by viewing the task of offloading from the mobile user's perspective. This is achieved by representing the job scheduling problem as a directed, acyclic graph. Figure 1 shows one such depiction where the root node is the user who wishes to offload some computational tasks to available vertices where the edges between vertices can be weighted to represent the energy or latency cost associated with selecting that vertex.

We adopt the popular partition/aggregate application structure which consists of a distributed set of aggregate and worker nodes [14]. Depending on the task partitioning, a user can forward data to the nearest aggregator that then schedules which worker nodes receive which jobs. Each aggregate node A_i can be seen as an independent job shop that receives a set of jobs J and in turn schedules them to be processed on a set of available worker machines W. In this paper, we focus on the first level of offloading from the user to the aggregator.

IV. KDN Based Offloading Framework

An overview of our solution framework is illustrated in Figure 2. The first step to our solution involves collecting network features for training an ML model. This part is important for developing the hyperprofile.

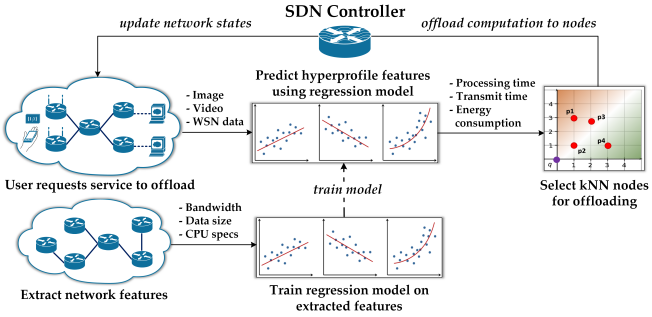

Fig. 2: Overview of our proposed offloading framework: after an offloading request, a pre-trained model is used to estimate features for a hyperprofile that can be queried using kNN to select destination nodes. The SDN controller serves to facilitate the offloading process.

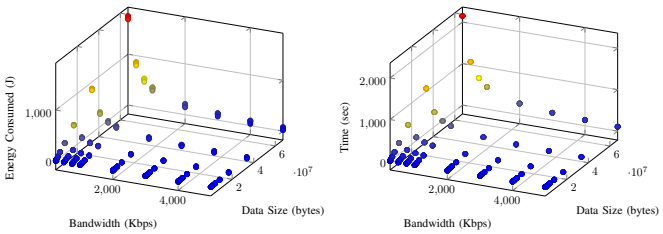

Fig. 3: ns-3 simulations: Energy consumption and data transfer time. The amount of data to send and bandwidth of the connection varied across simulations. Note the exponential relationship between energy/time and bandwidth for a fixed data size as well as the linear relationship between energy/time and data size for a fixed bandwidth.

A. Data Collection

We ran multiple simulations using ns-3 [15], a discrete-event network simulator that is available for research. Specifically, we ran our simulations using version 3.26 of ns-3 on Ubuntu 16.04 in parallel using GNU parallel [16]. We chose ns-3 as our network simulator for its tracing subsystem, energy framework for Wi-Fi devices, and ease of running the same simulation with modified program parameters. These capabilities allow us to generate training data. We simulated sending data between a wireless device and access point while varying bandwidth and total data sent. We measured energy consumed by the wireless device and the time that passed from the first packet being sent by the source to the last packet being received.

The simulations provide a basis for developing the hyperprofiles for the servers. Our goal is to predict energy consumption and transmission time from bandwidth and data size. Figure 3 depicts the relationship between energy consumption, transmission time, bandwidth, and data size as a scatter plot of the raw data.

B. Predictive Analytics

We found multi-step regression to be the most appropriate approach based on the properties of our training data. Bandwidth was used to predict the regression line that models energy consumption and data size. Our results show that the slope of such lines are exponential with respect to bandwidth. The same relationships apply to predicting time, as depicted in Figure 3.

Formally, we can represent the predicted variable as a linear function

$$f_b(d_s) = m(b)d_s + c(b)$$

where the slope $m(b)$ and the y-intercept $c(b)$ are functions of some bandwidth b. Our results show that m is exponential and can be defined accurately from historical data. Note that for energy consumption $c(b) = 0$ because the amount of energy consumed when sending 0 bytes of data will always be 0, regardless of bandwidth.

We perform linear regression for each fixed value of bandwidth between energy consumption and data size. In Figure 3 these lines can be seen by connecting scatter points along a particular bandwidth value. We predict the slope of these lines using bandwidth values. The results of these predictions are shown in Table I. The regression resulted in various curves of best fits which are given explicitly in the table.

Table I also reports the R^2 value of the models used to compute m and c. It also depicts the k-fold cross-validation score for $k = 10$. This score validates that for a fixed bandwidth the relationship between the predicted value and the data size can be modeled with a linear function.

	Energy Consumption (e_c)	Time (t)
Bandwidth (b)	$m_1 = 0.015b^{-1.13}$ $R^2 = 0.997$	$m_2 = {}^{8.04\cdot10^6}/_b$ $R^2 = 1$ $c = 222873e^{0.0004b}$ $R^2 = 0.918$
Data Size (d_s)	$e_c = m_1 d_s$ Cross-validation: 0.99	$t = m_2 d_s + c$ Cross-validation: 0.99

TABLE I: Accuracy of predictions. In all cases, the R^2 value is greater than 0.9. The lowest score comes from predicting the y-intercept of transmission time; to obtain a higher accuracy we could try collecting more data.

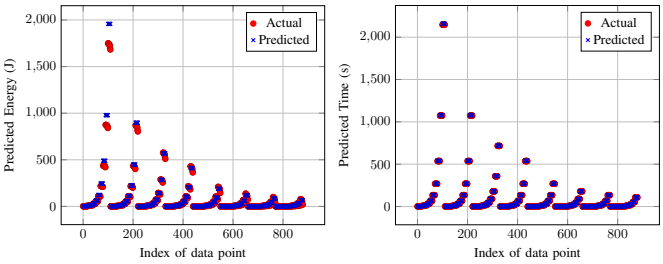

Fig. 4: Analysis of our prediction models. Each point on the x-axis represents a trial of a specific bandwidth and data size. The trials also varied physical distance, and even though our model did not account for distance, it still performed well – as shown by the overlap between the actual points and the predicted points.

In another set of simulations, we varied the physical distance between nodes from 10 m to 100 m. We avoided larger distances because we wanted to focus on scenarios where packet loss is not part of the issue of offloading. Testing our prediction model on this data shows that the variation of distances contributes an insignificant amount of error. The results of our predictions are given in Figure 4 where the x-axis plots the index of the data point and the y-axis represents dependent variable (both predicted and actual). The data clusters naturally into groups based on bandwidth, the first group being from index 0 to ~ 100. The energy consumption/transmission time increases within the groups because data size is increasing. Notice that the error is worst for smaller indices. This shows that the error is worst for small bandwidths and large data sizes. This observation could be attributed to the way error compounds with larger transmission times (since the largest transmission time occurs with a small bandwidth and a large data size). The main point is that in the vast majority of cases, we can almost exactly predict energy consumption and transmission time.

C. Hyperprofiles for Efficient Offloading

Our regression analysis shows that from historical data, we can develop accurate models of network features. A natural question is how we can leverage such models to make intelligent offloading decisions. Given that we are using ML to

predict network metrics specific to a server, an intuitive answer to this question involves representing the available servers in a "feature space" where each dimension of the space is a modeled metric (e.g. energy consumption). The user device (or aggregator) can then be intelligently placed in the feature space such that its position relative to the servers' positions conveys meaningful information. For our metrics, we want to minimize energy consumption and transmission time, so the user will always be represented by the origin. That way, distance from the origin conveys a level of desirability (i.e. the farther a node is from the origin the less desirable it is). This representation is particularly useful because if the device application needs to partition a task into k parts for computation offloading, it can perform a kNN query on the origin to get a set of server points in the feature space with the "optimal" resources for processing the task.

The metrics or "profiles" that represent available servers in the feature space do not necessarily have to be predicted values. They could be specifications of the servers themselves such as processor clock speed. To help distinguish between the various profiles, we introduce the following terminology.

Definition 1 (Base Profile)**.** A base profile for an edge network consists of points in a feature space where each point represents a unique server instance, and each dimension of the point represents a deterministic metric. Such metrics may include internal instance specifications (e.g. internal memory), characteristics of the network (e.g. bandwidth), or real-time metrics (e.g. CPU load).

Definition 2 (Hyperprofile)**.** A hyperprofile is similar to the base profile except each dimension represents a predicted metric. One example of such a metric is the estimated time to receive and transmit a data packet from an external host.

The main idea behind the different metrics is that they indicate a level of "fitness" of each server which can be quantitatively compared with the user device needs and specifications. Different profiles may be appropriate for different tasks and different scenarios. For example, in cases where no historical data is available one may opt to use the base profiles. An interesting direction for future research is to investigate the trade-off between the various profiles and whether one is significantly more useful than the others. It may even be helpful to combine the profiles into a *hybridprofile*. Regardless, the use of profiles is advantageous because it reduces the computation offloading problem to a kNN query.

D. Queries on hyperprofile features

We developed the idea of the hyperprofiles with the intention of performing kNN queries on the user device to obtain a set of server instances to which we offload. Nonetheless, it should be noted that different types of queries may employ different search algorithms. Moreover, it need not be the case that every dimension is a metric that we want to minimize. And hence, it may not be the case that the user is always represented by the origin.

Regardless, for our model, kNN was the most appropriate algorithm because it returns the points most "optimal" relative to the user. Formally, kNN returns a set of points for a query point q such that $p \in k\text{NN}(q)$ iff $|\{j \in P : d(j,q) < d(p,q)\}| < k$ where d is a predefined distance metric. Often the distance metric is Euclidean which means, in our case, $k\text{NN}(\vec{0})$ returns the points $p_i = (x_i, y_i)$ such that the values of $x_i^2 + y_i^2$ are minimal. When x is energy consumption and y is transmission time, offloading to the servers represented by points in $k\text{NN}(\vec{0})$ minimizes energy loss and latency. Other approaches (e.g. Chen [17]) minimize $x + y$ which is effectively the same as performing kNN where d is rectilinear distance. It may seem that minimizing one may be the same as minimizing the other, but as Table II depicts, this is not the case. The next section is dedicated to discussing the difference between these two distance metrics.

Point \ Distance metric	Euclidean	Rectilinear
$p_1 = (0.219, 0.371)$	0.431	0.59
$p_2 = (0.233, 0.361)$	0.429	0.594

TABLE II: Example scenario where a query of one point returns different sets. If using Euclidean distance $k\text{NN}(\vec{0}) = \{p_2\}$, whereas if using rectilinear distance, $k\text{NN}(\vec{0}) = \{p_1\}$. Notice that $|x - y|$ is smaller for p_2 than for p_1. This relationship is explored further by Proposition 1 (see Appendix).

V. Different distance metrics for kNN queries

We performed a set of simulations on randomly generated hyperprofiles to evaluate how often a kNN query would give different results for different distance metrics. Our simulations consisted of 2 dimensional hyperprofiles ranging in size from 250 points to 5000 points. Each point was computed from randomly generated bandwidth and data size values. Bandwidth ranged from 250 Kbps to 15 Mbps while data size ranged from 60 kB to 250 MB. For each hyperprofile we performed kNN queries where $k \in \{1, 2, 3, 4, 5, 10\}$. The queries were performed with both Euclidean and rectilinear metrics and the mismatches between the two methods is shown in Figure 5.

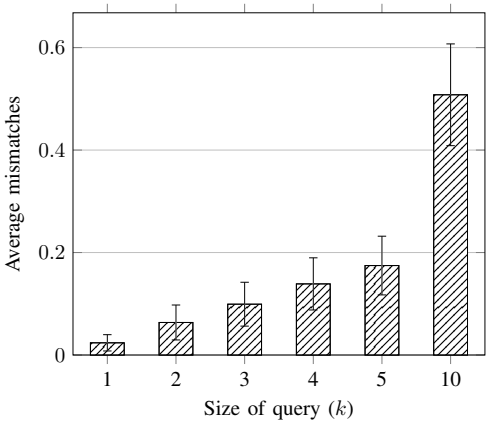

Fig. 5: Average mismatches between queries with 95% confidence intervals. Size of the dataset and queries were varied across simulations. Queries were performed on a dataset of randomly generated bandwidth and data size values. Note larger queries have a higher probability of mismatch.

Throughout the simulations, we kept track of the points that were mismatched, and in all the cases we noticed that the Euclidean distance queries returned points with minimal differences between x and y. In other words, it returned points

closer to the line $y = x$. After formalizing our observation, we found this was an inherent property of the mismatched points. Thus, as Proposition 1 (See Appendix) states, if there is a mismatch in the minimal Euclidean distance and the minimal rectilinear distance between points then the distance between the coordinates of the minimal point based on Euclidean distance is less than the distance between the coordinates of the minimal point based on rectilinear difference.

Notice that the first condition of the proposition is that the coordinates are nonnegative. Without this condition, a simple counterexample such as $(x_1, y_1) = (3, 0)$ and $(x_2, y_2) = (-3, 1)$ would falsify the proposition. The condition is reasonable since often the hyperprofile dimensions represent nonnegative features of the network or user device such as energy consumption or latency.

Ultimately, the key point of Proposition 1 is that a Euclidean distance metric will favor points with a more balanced tradeoff between the network features represented by the coordinates. Moreover, the difference becomes more pronounced as k becomes larger. Hence, whether this tradeoff is favorable depends on both the size of the edge network and the types of features with which the user is concerned.

VI. Conclusion

In this paper, we outlined a framework for computation offloading in disaster response networks by creating a hyperprofile of predicted resource consumption for edge nodes based on metrics such as bandwidth and data size. Our solution is unique because it can be easily adapted to different metrics based on a user's needs. This is accomplished by decoupling the problem of modeling the available network metrics from the problem of identifying user needs while making use of machine learning models. Effectively, we have described a way of *encoding* available server instances in a multi-dimensional space to facilitate effective queries. For future work, we plan to implement a testbed to compare a hyperprofile-based offloading scheme to standard offloading schemes. Another area of interest is expanding the concept of a hyperprofile to other edge network areas such as routing or trust management.

Appendix

Proposition 1. *If* (1) $x_1, y_1, x_2, y_2 \geq 0$, (2) $x_1^2 + y_1^2 < x_2^2 + y_2^2$, *and* (3) $x_2 + y_2 < x_1 + y_1$ *then* $|x_1 - y_1| < |x_2 - y_2|$.

Proof. First note that $x_2 \neq x_1$ since if they were equal then we would have $y_1^2 < y_2^2$ and $y_2 < y_1$, a contradiction. The same reasoning implies $y_2 \neq y_1$. Now assume without loss of generality that $x_1 \geq y_1$. We can write (3) as

$$(x_2 - x_1) + (y_2 - y_1) < 0.$$

We deal with the problem in cases based on whether $(x_2 - x_1)$ or $(y_2 - y_1)$ is negative. For the first case, assume $x_2 - x_1 < 0$. Now write (2) as

$$(y_1 - y_2)(y_1 + y_2) < (x_2 - x_1)(x_2 + x_1).$$

Since $x_2 - x_1 < 0$ and the sums are positive, we must have $y_1 - y_2 < 0$. Hence, $y_2 - y_1 > 0$. Now write (2) as

$$(x_1 - x_2)(x_1 + x_2) < (y_2 - y_1)(y_2 + y_1). \quad (1)$$

and write (3) as $y_2 - y_1 < x_1 - x_2$. Combining these we get

$$(x_1 - x_2)(x_1 + x_2) < (x_1 - x_2)(y_2 + y_1).$$

Since $x_1 - x_2$ is positive we can divide it out to get $x_1 + x_2 < y_2 + y_1$ which we rearranging to get,

$$|x_1 - y_1| = x_1 - y_1 < y_2 - x_2 \leq |y_2 - x_2|$$

by our original assumption that $x_1 \geq y_1$. For case (2) assume that $(y_2 - y_1) < 0$. From Equation (1), this means $x_1 - x_2 < 0$. Now combining these with our assumption that $x_1 \geq y_1$ we have $x_2 > x_1 \geq y_1 > y_2 \geq 0$. From this, we can write

$$|x_2 - y_2| \geq x_2 - y_2 > x_1 - y_1 = |x_1 - y_1|.$$

□

Acknowledgements

This material is based upon work supported by the National Science Foundation under Award Number: CNS-1659134 [REU Site], CNS-1647182, and CNS-1647084. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] R. Gargees *et al.*, "Incident-supporting visual cloud computing utilizing software-defined networking," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 1, pp. 182–197, 2017.

[2] X. Ma *et al.*, "When mobile terminals meet the cloud: computation offloading as the bridge," *IEEE Network*, vol. 27, no. 5, pp. 28–33, 2013.

[3] F. Samie *et al.*, "Computation offloading and resource allocation for low-power iot edge devices," *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, p. 712, 2016.

[4] H. Eom *et al.*, "Malmos: Machine learning-based mobile offloading scheduler with online training," in *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2015 3rd IEEE International Conference on*. IEEE, 2015, pp. 51–60.

[5] E. Cuervo *et al.*, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 49–62.

[6] B.-G. Chun *et al.*, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 301–314.

[7] X. Gu *et al.*, "Adaptive offloading for pervasive computing," *IEEE Pervasive Computing*, vol. 3, no. 3, pp. 66–73, 2004.

[8] A. Mestres *et al.*, "Knowledge-defined networking," *arXiv preprint arXiv:1606.06222*, 2016.

[9] D. D. Clark *et al.*, "A knowledge plane for the internet," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '03. ACM, 2003, pp. 3–10.

[10] H. Trinh *et al.*, "Energy-aware mobile edge visual data processing," *To Appear in IEEE International Conference on Future Internet of Things and Cloud (FiCloud)*, 2017.

[11] M. E. Khoda *et al.*, "Efficient computation offloading decision in mobile cloud computing over 5g network," *Mobile Networks and Applications*, vol. 21, no. 5, pp. 777–792, 2016.

[12] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 4th ed. Springer, 2012.

[13] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on computing*, vol. 3, no. 2, pp. 149–156, 1991.

[14] M. A. Attar, "Packet transport mechanisms for data center networks," Ph.D. dissertation, Stanford University, 2013.

[15] "ns-3." [Online]. Available: www.nsnam.org

[16] O. Tange, "Gnu parallel - the command-line power tool," *;login: The USENIX Magazine*, vol. 36, no. 1, pp. 42–47, Feb 2011. [Online]. Available: http://www.gnu.org/s/parallel

[17] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, 2015.