By Aditya Khune
and Sudeep Pasricha

Using reinforcement learning to make reward-based decisions in smartphone applications.

Mobile Network-Aware Middleware Framework for Cloud Offloading

Offloading mobile computations is an innovative technique to reduce energy consumption in mobile devices and minimize application response time. In this article, we propose a middleware framework that uses reinforcement learning (RL) to make reward-based offloading decisions. Our framework allows a smartphone to consider suitable contextual information to determine when it makes sense to offload and select between available networks when offloading. We tested our framework in simulated and real environments, across various apps, to demonstrate how energy consumption can be minimized in mobile devices capable of supporting offloading to the cloud.

Digital Object Identifier 10.1109/MCE.2018.2867972
Date of publication: 10 December 2018

2162-2248/19©2019IEEE

©ISTOCKPHOTO.COM/MUSTAFAHACALAKI

DEVELOPING OFFLOADING OPPORTUNITIES

Faster wireless network speeds and rapid innovations in mobile technologies have changed the way we use our computers. It is estimated that 207.2 million people in the United States own a smartphone today, whereas the number of smartphone users worldwide is estimated to be more than 2 billion [1]. The volume of data being accessed and processed by smartphones and the sophistication of mobile apps is rapidly increasing over time. However, the evolution in hardware and software capabilities of mobile devices has not been paralleled by similar advances in battery technology. Today, high-end mobile apps increasingly burden the limited battery life of smartphones. For example, a GPS app can drain a phone's battery completely within 7 h [2].

A promising solution that is being considered to support high-end mobile apps is to offload mobile computations to the cloud [3]–[9]. Offloading is an opportunistic process that relies on cloud servers to execute the functionality of an app that typically runs on a mobile device. Such computation offloading is being considered today as a means to save energy consumption (thereby improving battery lifetime) and increase the responsiveness of mobile apps.

We propose a novel middleware framework that uses a machine-learning technique called *RL* to make offloading decisions effectively on a smartphone. The proposed framework considers various types of information on the mobile device, such as the network type, the network bandwidth, user context, and so on, to decide when to offload to minimize energy consumption. Our strategy selects between available networks [third generation (3G), fourth generation (4G), or Wi-Fi] when offloading mode is active. Our experiments with real apps on a smartphone highlight the potential of our framework to minimize energy consumption.

CHALLENGES WITH OFFLOADING

In spite of existing research highlighting the potential of offloading in mobile devices, current offloading techniques are far from being widely adopted in mobile systems. The implementation of these computation offloading techniques for many real-world mobile apps in real-world scenarios has not shown promising results [3], with the mobile device consuming more energy in the offloading process than the energy savings achieved due to computing on servers in an offloaded manner.

Offloading decision engines must consider not only the potential energy savings from offloading but also how the response time of the app is impacted by offloading. An effective decision to offload processing to the cloud must reduce energy without significantly increasing response time. Such decisions are heavily impacted by wireless network inconsistency. The power consumed by the network radio interface is known to contribute a considerable fraction of the total device power, and it varies depending on wireless signal strength [10]. With the recent advent of high-bandwidth 4G networks, there has been increased interest in the offloading domain, but from our experiments, we found that the 4G network consumes more energy than Wi-Fi and the 3G network.

The network quality of a 4G connection at a mobile device's location greatly affects the battery life. If the device is in the area that does not have 4G coverage, there is no advantage to a 4G interface, and if the 4G network search is not disabled, then the radio's search for a nonexistent signal will drain the battery quickly. In the case of a weak signal, the device uses more power to send and receive data to and from the network. A strong 4G signal uses less battery, but the biggest problem is the constant switching from the 4G network to the 3G network and back again. Also, throughout a typical

day, at different times, the performance of a wireless network varies because of changing traffic load on the network. We refer to all such problems due to the mobile network as "network inconsistency" problems.

OFFLOADING PERFORMANCE OF MOBILE APPS

We analyzed the performance implications of offloading by comparing two scenarios—one where all computations are performed only on the mobile device without using the cloud at all (local mode) and the other where there was a complete reliance on the cloud computation (offload mode), with minimal computations on the mobile device. We selected five diverse commercially available smartphone apps for our experiments: 1) matrix operations; 2) Internet browser; 3) zipper (file compression); 4) voice recognition/translation; and 5) torrent (file download). The experiments were performed on an LG G3 device running the Android OS version 5.0.1 around the Colorado State University campus in Fort Collins.

Figure 1 shows the results of our experiments for the torrent app (results for the other apps are omitted for brevity). We used the Android-based torrent app Flud [11] to perform torrent downloads in the local mode. In the cloud mode, a cloud server is used as a BitTorrent client to download torrent pieces on behalf of a mobile device. While the cloud server downloads the torrent, the mobile device switches to the sleep mode until the cloud finishes the torrent processes, and then the cloud uploads the downloaded torrent file in a single process to the mobile device. For our experiments, we used torrent file sizes ranging from 25 to 85 MB, with an Amazon Web Services cloud instance being used for the cloud mode [12]. Of all five apps that we analyzed, offloaded processing proves to be most beneficial in terms of both energy savings and response time for the torrent download app, which is data intensive but not compute intensive. The 4G network is faster than the 3G network but slower than Wi-Fi. The 4G network performs slightly better than the 3G network in terms of energy consumption for higher data sizes (45–85 MB), but for smaller data sizes, the 3G network is more energy efficient.

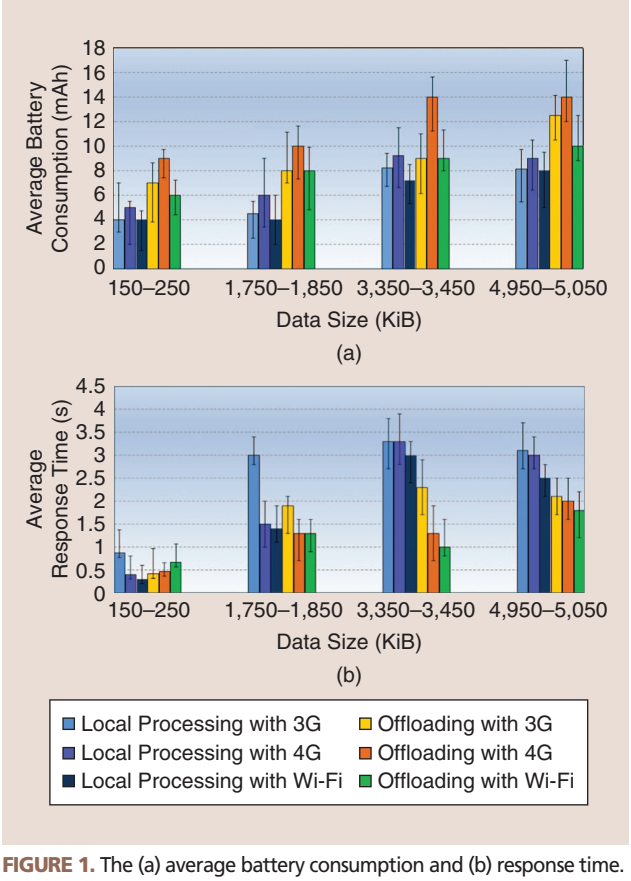

FIGURE 1. The (a) average battery consumption and (b) response time.

SUMMARY OF FINDINGS

The overall performance when offloading depends on various factors such as the amount of data required by the app, the wireless network signal type and strength, and the functionality of the app under consideration. In some prior work [13], it was concluded that offloading is beneficial when an app is compute intensive and at the same time less data intensive. However, we found that this is not always the case. For instance, offloading is beneficial for apps that may not be compute intensive, but are data intensive, e.g., the torrent app. To make offloading more practical, it is important to reduce the energy spent in the communication between the mobile device and the cloud.

Our experiments indicate that choosing the best possible network for offloading is a critical decision. One may assume that because the 4G network is faster than the 3G network, we should always rely on it for offloading when Wi-Fi is not available. However, our results indicate that the 4G network is more power hungry than the 3G network most of the time. Network quality is also a factor that cannot be ignored. We found that a good 3G coverage performs far better as opposed to poor 4G coverage, and vice versa. In the region of cell tower edges or where the coverage of 3G/4G ends, we found that the handover process results in high battery drain. This is because the device in such scenarios is constantly searching for the network, frequently scanning the wireless spectrum around it to determine to which tower it should tether itself. The more networks there are to choose from, the longer the scans take. Some apps require a channel to be established between the base station and the mobile device at regular intervals, which can significantly drain the device battery.

ADAPTIVE OFFLOADING

The decision to offload a mobile app to the cloud is a complex one due to the distributed nature and many real-time constraints of the overall system. To make an effective offloading decision, it is vital to consider various factors, as we discovered after our experimental analysis from the previous section. As these factors vary at run-time, there is a need for an adaptive offloading approach that takes the variations of these factors at run-time into consideration when making decisions.

Flores and Srirama [4] proposed a fuzzy decision engine for code offloading that considers the contextual parameters on a device and the cloud to make an offloading decision adaptively.

The mobile device runs the fuzzy logic decision engine, which is used to combine *n* number of variables (e.g., app data size, network bandwidth) that are obtained from the overall mobile cloud architecture. The fuzzy logic decision engine works in three steps: fuzzification, inference, and defuzzification. In fuzzification, input data are converted into linguistic variables, which are assigned to a specific membership function. A reasoning engine is applied to the variables, which makes an inference based on a set of rules.

Finally, the outputs from the reasoning engine are mapped to linguistic variable sets again in the defuzzification step. This offloading decision engine in [4] assumes a consistent network performance during offloading. However, as observed in our experiments, such consistency is difficult to achieve because of frequent mobile user movements and variable network quality (due to factors such as the location of the device and the load on the network). Moreover, the offloading decision engine mainly emphasizes energy savings; however, the response time is also a crucial metric for various apps that should not be ignored; otherwise, the user quality of service degradation can become so severe that any effort to save energy becomes irrelevant.

MIDDLEWARE FRAMEWORK FOR EFFICIENT OFFLOADING OF MOBILE APPS

To simplify the mobile app development process and at the same time avoid problems caused by hard-coded annotations, our framework proposes to transfer all the computation for an app to the cloud instead of partial (selective) offloading of the app. Our framework involves a novel decision engine on the mobile device that works together with a clone virtual machine of the mobile software environment to execute apps on cloud servers.

> Q-learning is a reward-based mechanism that generates a Q-table with reinforcement or penalty values.

Figure 2 shows a high-level overview of the proposed framework. Our framework is implemented at the middleware level and runs in the background as an Android service. As a result, it requires no changes to any of the apps or the Android OS. The runtime monitor component periodically triggers the RL module to generate/update a Q-learning table. At any time, this Q-table contains information to guide the decision for when and how to offload an app to the cloud, depending on multiple factors.

RL is an unsupervised learning approach that focuses on learning by having software agents interact with an environment and then taking actions to maximize some notion of a reward. In supervised learning (e.g., using neural networks), a training set of correctly identified observations is required to train a prediction model. RL differs from supervised learning in that correct input/output pairs of identified observations do not need to be presented, so there is no need for a pretrained model.

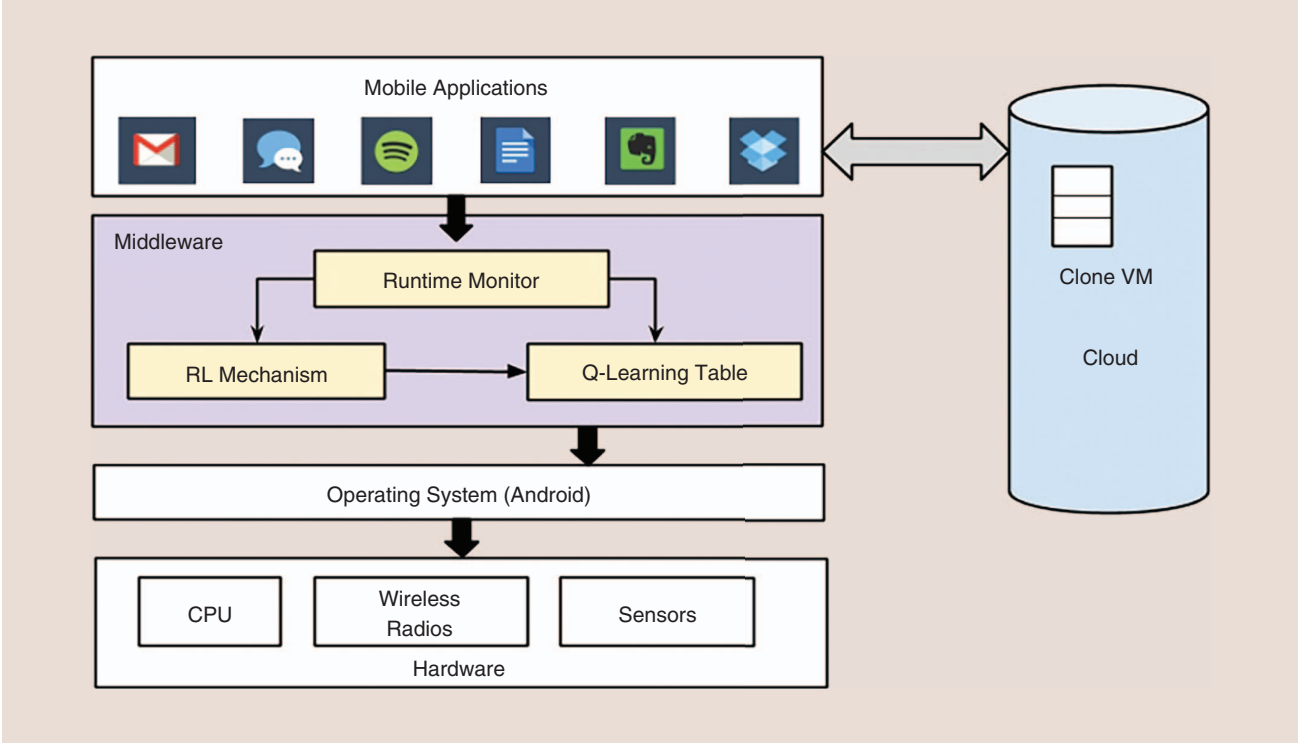

FIGURE 2. The RL-based middleware framework for efficient application offloading to the cloud. VM: virtual machine; CPU: central processing unit.

In RL, the state-action value function is a function of both state and action, and its value is a prediction of the expected sum of future reinforcements. The state-action value function is referred to as the *Q-function*. Figure 3 summarizes how a typical Q-learning reinforcement algorithm works. Q-learning is a reward-based mechanism that generates a Q-table with reinforcement or penalty values. The figure illustrates a section of a Q-table where the possible actions are offloading with the 3G, 4G, or Wi-Fi network, when the user is at different locations L1–L4. Actions are chosen, and the penalty values are calculated for respective actions to update the Q-table. Suppose the system is at a defined state S_t at time t. After taking action a_t from that state, we observe the one-step reinforcement r_{t+1}, and the next state becomes s_{t+1}. This continues until we reach a goal state, K, steps later. The objective with RL is to find actions a_t that maximize (or minimize) the sum of reinforcements or rewards r_t. This can be reduced to the objective of acquiring the Q-function that predicts the expected sum of future reinforcements; where the correct Q-function determines the optimal next action.

In our problem formulation, the state of a mobile device is defined using the contextual information of the device such as its location, available network type, and network strength. These contextual factors are chosen as we consider them to be crucial for efficient offloading. The runtime monitor extracts the contextual information of the device to form state values of the system. For example, consider a mobile device that is at location L1, where it has access to a 3G network type with "strong" network strength. From this state, if an app processing needs to be offloaded, then the Q-function is called to select the appropriate network that would result in the least penalty in terms of energy or response time (or both). In our framework, the following state and action values are used to generate the Q-function:

- set of state values (discrete values):
  - location = L1, L2, L3,…., Ln
  - network carrier = 3G, 4G, Wi-Fi
  - network strength = strong, medium, weak
  - data size = small, medium, large.
- set of action values:
  - offload using the 3G network
  - offload using the 4G network
  - offload using the Wi-Fi network.

The location L1–Ln can be any geographic area where the user utilizes the offloading app, e.g., office, home, and so on. More state-action pairs can be added to the aforementioned list to account for factors that might affect offloading, e.g., we can add "time of day" as another state value, as it is observed that network performance is slow at certain times of the day when the network load is high. However, a larger set of state-value pairs will result in a larger Q-function requiring greater overhead to manage it.

The Q-function is generated as follows. Initially, when the mobile device is at location L1, the runtime monitor accesses contextual information from the device such as location, networks available, and network strength. A small data file is then uploaded from the mobile device to the cloud, using the primary network carrier. The amount of battery consumed and the total response time taken for this operation are measured. The uploading operation is repeated with varying (small, medium, and large) data sizes with all available networks at the location (3G, 4G, and Wi-Fi) activated one by one. For each of these uploading operations the runtime monitor measures the battery amount consumed and response time to complete the operation. The Q-table is then populated with the penalty values calculated using

$$P_{3G} = P_{b3G} * x + P_{t3G} * y, \tag{1}$$

$$P_{4G} = P_{b4G} * x + P_{t4G} * y, \tag{2}$$

$$P_{\text{WiFi}} = P_{\text{WiFi}} * x + P_{\text{WiFi}} * y. \tag{3}$$

Thus, in our RL framework, the reinforcement values are essentially the penalty values P_{3G}, P_{4G}, and P_{WiFi}. Once

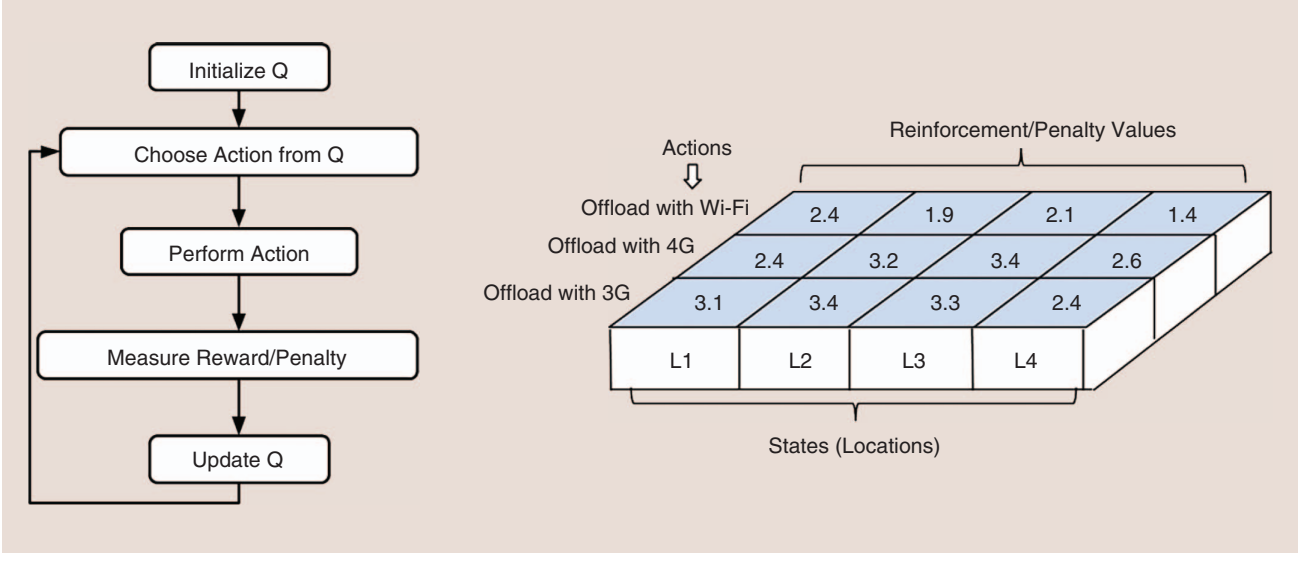

FIGURE 3. The Q-learning flow with an example of a Q-table.

populated, the Q-table can be updated periodically in the background when the user is not actively using the device. In (1)–(3), to optimize battery consumption and response time, we used weights x and y, respectively with penalty values. We used $x = y = 0.5$ to balance minimizing battery consumption and response time.

EXPERIMENTAL RESULTS

To evaluate the efficacy of our proposed framework, we conducted a set of experiments. We implemented our middleware framework and its decision engine on an Android-based mobile device. To form the Q-function of our RL algorithm, real user data were collected at different geographical locations around the Colorado State University campus area. We compared our work with the fuzzy logic decision engine proposed by Flores and Srirama [4], which we discussed previously and which we implemented on the Android-based mobile device.

Figure 4 shows the results for the matrix operation app with our proposed RL-based decision engine and the fuzzy logic-based decision engine from [4]. Similarly, Figure 5 includes the results for the zipper app, and Figure 6 shows results for the Torrent app. In all the scenarios, the task of a decision engine is to decide whether to offload and to select the network to offload with. In these figures, the red trendline shows results with the fuzzy decision engine [4] whereas the green trendline shows the results with our RL-based middleware framework. We have also shown bars with the results for offloading with each available network and local processing as a reference.

In general, our results reveal that our proposed RL-based decision engine outperforms the fuzzy logic approach from [4]. For less data-intensive operations, the results of RL and fuzzy logic overlap. For instance, in the case of the zipper app (Figure 5), for lower data sizes fuzzy logic shows better results, because the Q-table generated using our RL algorithm uses 25 MB as the minimum data size. For any data size lower than this minimum value, the RL-based framework is thus less effective at making predictions. This can be improved using a wider range of data files/sizes when populating the Q-table. For higher data sizes and more complex computations,

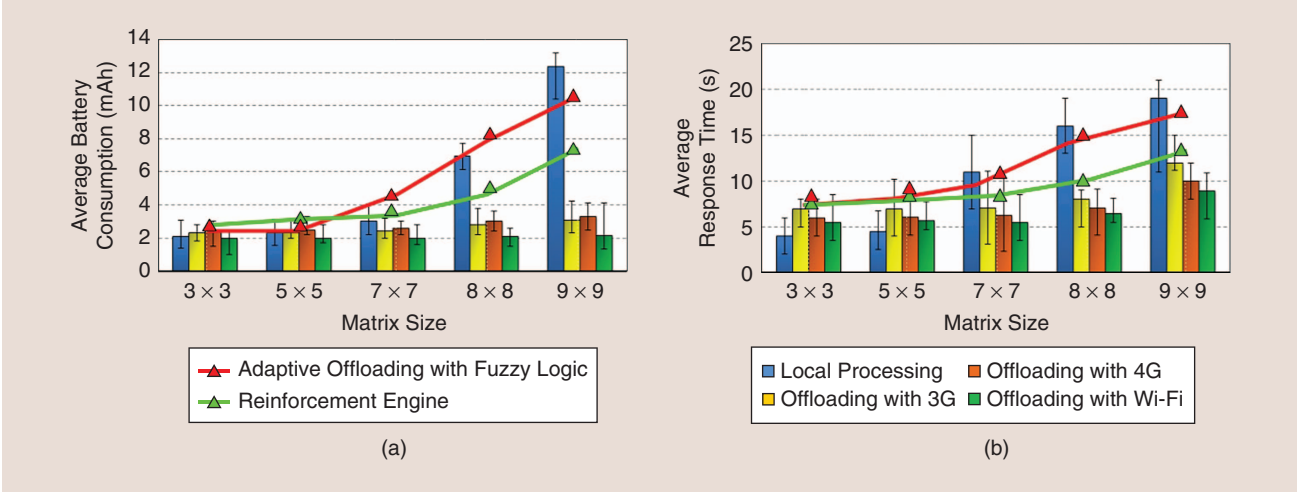

FIGURE 4. The (a) average battery consumption and (b) response time of a matrix operations app with learning methods.

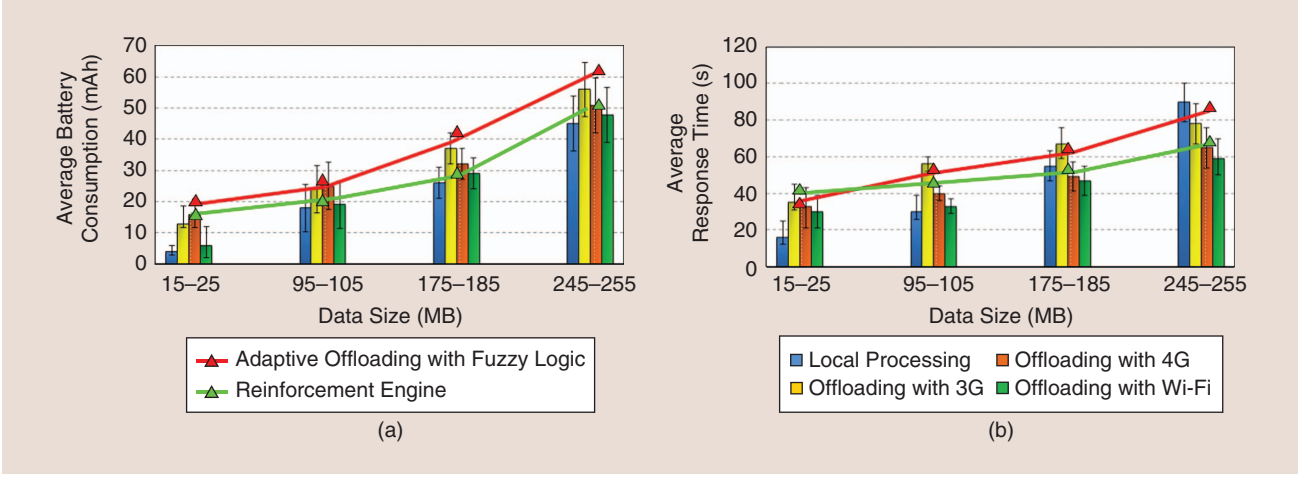

FIGURE 5. The (a) average battery consumption and (b) response time of the zipper app with learning methods.

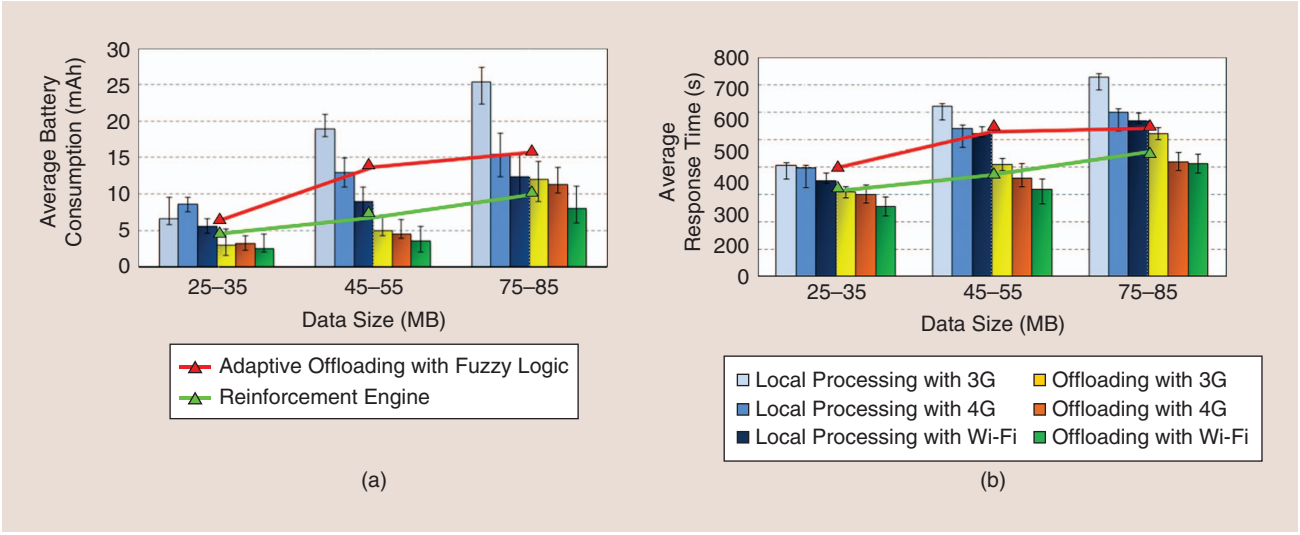

FIGURE 6. The (a) average battery consumption and (b) response time of the torrent app with learning methods.

our RL approach gives an improved battery consumption and response time than [4].

Our RL-based engine has better prediction accuracy that is crucial for making effective offloading decisions. The overall performance of offloading depends on various factors, such as the amount of data required by the app, the network signal type (3G, 4G, and Wi-Fi) and network signal strength, and the complexity of the functionality of the app under observation. By considering all of these individual factors in the decision process, unlike the fuzzy logic approach from [4], and by using a more sophisticated and powerful learning algorithm, our framework is able to achieve notably better results compared with [4]. Our results show that the proposed RL-based offloading system can save up to 30% battery power with up to 25% better response time compared with the fuzzy logic-based approach.

CONCLUSION

We analyzed real mobile apps to determine the benefits of app offloading and presented a novel network-aware mobile middleware framework based on RL to accomplish energy-efficient offloading in smartphones. Our results show that we can save up to 30% battery power with up to a 25% better response time when using our proposed framework compared with a state-of-the-art fuzzy logic-based offloading approach from prior work.

ACKNOWLEDGMENT

This work is supported in part by grants from the National Science Foundation (ECCS-1646562, CCF-1252500).

ABOUT THE AUTHORS

Aditya Khune (adkhune@colostate.edu) is a software engineer with Dell EMC in Louisville, Colorado.

Sudeep Pasricha (sudeep@colostate.edu) is a Monfort and Rockwell-Anderson professor of electrical and computer engineering at Colorado State University, Fort Collins.

REFERENCES

[1] The Statistical Portal. (2016). [Online]. Available: www.statista.com/statistics.

[2] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt, "Micro-blog: Sharing and querying content through mobile phones and social participation," in *Proc. ACM Mobisys*, 2008, pp. 174–186.

[3] H. Flores and S. Srirama, "Mobile code offloading: Should it be a local decision or global inference?" in *Proc. ACM Mobisys*, 2013, pp. 539–540.

[4] H. R. Flores and S. Srirama, "Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning," in *Proc. ACM Mobisys*, 2013, pp. 9–16.

[5] G. Banga, S. Crosby, and I. Pratt, "Trustworthy computing for the cloud-mobile era: A leap forward in systems architecture," *IEEE Consum. Electron. Mag.*, vol. 3, no. 4, pp. 31–39, Oct. 2014.

[6] P. Corcoran and S. K. Datta, "Mobile-edge computing and the Internet of things for consumers: Extending cloud computing and services to the edge of the network," *IEEE Consum. Electron. Mag.*, vol. 5, no. 4, pp. 73–74, Oct. 2016.

[7] H. Flores and S. Srirama, "Mobile code offloading: From concept to practice and beyond," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 80–88, Mar. 2015.

[8] B. Donohoo, C. Ohlsen, S. Pasricha, C. Anderson, and Y. Xiang, "Context-aware energy enhancements for smart mobile devices," *IEEE Trans. on Mobile Computing*, vol. 13, no. 8, pp. 1720–1732, Aug. 2014.

[9] M. E. Khoda, M. A. Razzaque, A. Almogren, M. M. Hassan, A. Alamri, and A. Alelaiwi, "Efficient computation offloading decision in mobile cloud computing over 5G network," *Mobile Netw. Applicat.*, vol. 21, no. 5, pp. 777–792, Oct. 2016.

[10] J. Li, K. Bu, X. Liu, and B. Xiao, "Enda: Embracing network inconsistency for dynamic application offloading in mobile cloud computing," in *Proc. ACM Special Interest Group on Data Communications*, 2013, pp. 39–44.

[11] Fuld–Torrent Downloader. (2016). [Online]. Available: https://play.google.com/store/ apps/details?id=com.delphicoder.flud&hl=e

[12] Amazon Web Services (AWS). (2016). [Online], Available: https://aws.amazon.com/

[13] M. Altamimi, A. Abdrabou, K. Naik, and A. Nayak, "Energy cost models of smartphones for task offloading to the cloud," *IEEE Trans. Emerg. Topics Comput*, vol. 3, no. 3, pp. 384–398, Sept. 2015.