Quantifying the Impact of Edge Computing on Mobile Applications

Wenlu Hu, Ying Gao, Kiryong Ha, Junjue Wang, Brandon Amos, Zhuo Chen,
Padmanabhan Pillai†, Mahadev Satyanarayanan

Carnegie Mellon University and †Intel Labs

ABSTRACT

Computational offloading services at the edge of the Internet for mobile devices are becoming a reality. Using a wide range of mobile applications, we explore how such infrastructure improves latency and energy consumption relative to the cloud. We present experimental results from WiFi and 4G LTE networks that confirm substantial wins from edge computing for highly interactive mobile applications.

1. Introduction

Edge computing is a new paradigm in which substantial compute and storage resources are placed at the edge of the Internet, in close proximity to mobile devices or sensors. Terms such as "cloudlets" (the term we use in this paper) [22], "micro data centers (MDCs)" [12], "fog" [1], and "mobile edge computing (MEC)" [4] are used to refer to these small, edge-located computing nodes. Edge computing is motivated by its potential to improve scalability, latency, and bandwidth over a cloud-only model. More practically, some efforts stem from the drive towards software-defined networking (SDN) and network function virtualization (NFV), and the fact that the same hardware can provide SDN, NFV, and edge computing services. This suggests that infrastructure providing edge computing services may soon become ubiquitous, and may be deployed at greater densities than content delivery network (CDN) nodes today. However, despite the momentum in this space, it is unclear how much such systems actually benefit end users. Our goal in this paper is to provide quantitative answers to this question.

We focus on a specific benefit of edge computing, namely the ability to offload computation at low latency from a mobile device to a cloudlet. This capability was first demonstrated in 1997 by Noble et al. [19], who used it to implement speech recognition with acceptable performance on a resource-limited mobile device. In 1999, Flinn et al. [8] extended this approach to improve battery life. These concepts were generalized in a 2001 paper that introduced the term *cyber foraging* for the amplification of a mobile device's data or compute capabilities by leveraging nearby infrastructure [20]. Today, offloading to the cloud is mainstream. It is used for speech recognition on both iOS and Android devices. For future applications that are both compute- and latency-sensitive, such as cognitive assistance [5] or mobile augmented reality, offloading to cloudlets at the network edge may be the new norm.

In this paper, we ask "How much better is it to offload to a cloudlet rather than to the cloud?" We focus on metrics that are important to end users, namely response time and energy consumption. On a variety of applications, some prepartitioned by a developer and others dynamically partitioned by a runtime system, we quantify how cloudlets impact these metrics relative to cloud offload. We explore how offloading over WiFi and 4G LTE differ in these metrics. We also report on the sensitivity of these metrics to the interactivity level of an application.

2. Experimental Approach

Studying mobile offload to cloud and cloudlets is complicated by two factors. First, most existing applications have been tuned to run well in the mobile or mobile-plus-cloud environment. They have been designed around the resource limits of the mobile device and relatively high latency to the cloud. Hence, there are no production-quality applications that are too processing intensive to run on a mobile device but too latency sensitive for the cloud. Since this combination of application characteristics is exactly the sweet spot for edge computing, we have scoured the literature and have identified research applications with these characteristics (Section 2.1). These are statically prepartitioned into cloud and mobile device components.

Secondly, the benefits of offloading depend heavily on the quality of the partitioning of the applications. Static prepartitioning may not be optimal. When resources such as wireless network bandwidth vary, dynamic partitioning

APSys'16, August 04-05, 2016, Hong Kong, China.

ACM ISBN 978-1-4503-4265-0/16/08.

DOI: http://dx.doi.org/10.1145/2967360.2967369

Application	Request size (avg)	Response size (avg)
FACE	62 KB	< 60 bytes
MAR	26 KB	< 20 bytes
FLUID	16 bytes	25 KB

Figure 1: Network load of prepartitioned apps

Application	Total Transfer Size	# of Transfers †
Linpack	≈ 10 MB	1
CPU Benchmark	≈ 80 KB	1
PI Benchmark	≈ 10 MB	15

† Number of thread migrations for each run

Figure 2: Network load of COMET apps

may be necessary. As an unbiased external factor in our results, we use an existing tool called COMET [11] that represents the state of the art in dynamically partitioning off-the-shelf Android applications. It can transparently migrate threads from a mobile device to a remote server and back, guided by its dynamic partitioning algorithm. For the same reasons as for the prepartitioned applications, there are no compute-intensive latency-sensitive applications available off the shelf. So we use compute-intensive benchmarks, with a preference for the ones used in the original COMET paper, to represent this kind of application.

By exploring a diversity of data points, we increase the likelihood of our insights being robust and significant. With emphasis on response time as the key indicator of performance, we address the following issues relative to the choice of offloading site:

- effect on prepartitioned applications.
- effect on COMET-partitioned applications.
- interplay with an application's interactivity level.
- interplay with the type of wireless network.
- effect on the energy usage of a mobile device.

2.1 Applications

We study three statically prepartitioned applications from the existing research literature:

- FACE is a face recognition application that first detects faces in an image, and then attempts to identify the face from a small, pre-populated database. It implements the Eigenfaces method [25] using OpenCV [3], and runs on Microsoft Windows.
- MAR is an augmented reality application that labels buildings and landmarks in a scene [24]. The prototype application uses a dataset of 1005 labeled images of 200 buildings. MAR runs on Microsoft Windows, and makes significant use of OpenCV [3], Intel Integrated Performance Primitives (IPP) [15], and multiple processing threads.
- FLUID is an example of physics-based computer graphics [23]. The Linux application runs a multithreaded physics simulation of an imaginary fluid and uses client accelerometer readings to let users interact with the model. We run FLUID on a 2218-particle system with 20 ms timesteps, generating up to 50 frames per second.

Each of these applications is split into a front-end mobile app and a back-end server that performs most of the computations. The mobile front-end captures and sends inputs (images for FACE and MAR, accelerometer values for FLUID), and shows the outputs (displaying text labels for FACE and MAR, and rendering a set of particles for FLUID). The differences in the inputs and outputs are reflected in the network transfer sizes for these applications, shown in Figure 1.

We also investigate three unmodified Android apps using COMET. One of them is from the original COMET paper, and the other two are from the Google Play store:

- Linpack for Android [6] performs numerical linear algebra computations to solve a dense N-by-N system of linear equations. This benchmark is used in the original COMET paper [11].
- CPU Benchmark [26] is a simple benchmarking tool designed for testing and comparing phone processing speeds and effects of CPU overclocking.
- PI Benchmark [17] measures CPU and memory speed by calculating π with up to 2 million digits of precision. It periodically saves intermediate state to a file, possibly triggering thread migration by COMET.

All of these applications are compute-intensive, and should benefit greatly from offloading. However, as shown in Figure 2, they differ in the amount and frequency of data transfers when run with COMET.

2.2 Experimental Setup

Figures 3 and 4 characterize the cloudlet, cloud, and mobile devices we use for the experiments. In the cloud (Amazon EC2), we use the most powerful virtual machine (VM) instance available to us in terms of CPU clock speed. The instance has 8 cores (VCPUs), each with the fastest available clock rate (2.8 GHz); its memory size is more than enough for all of the tested applications (15 GB). For cloudlet offload, we use a VM running on a Dell Optiplex 9010 desktop machine with 4 cores (VCPUs), whose clock is limited to 2.7 GHz. The VM instance is allocated 4 GB of RAM. By comparing a relatively weak cloudlet against more powerful EC2 cloud instances, we have deliberately stacked the deck against cloudlets. Hence, any cloudlet wins in our experiments should be considered quite meaningful.

We use two different mobile devices in our experiments. Since COMET does not run on Android versions beyond 4.1.X, those experiments use a Samsung Galaxy Nexus phone that runs Android 4.1.1. For the prepartitioned applications, our choice of hardware is limited by the fact that some of our applications only run on x86 hardware. Hence, all experiments with prepartitioned applications use a Dell Latitude 2120 netbook whose computational power is comparable to the latest smartphones.

Li et al. [16] report that average round trip time (RTT) from 260 global vantage points to their optimal Amazon EC2 instances is 74 ms, which is similar to the RTTs for EC2-West in our experimental setting. We note that our

Cloudlet VM on Dell Optiplex 9010	Cloud (Amazon AWS) c3.2xlarge instance
Intel® Core® i7-3770 2.7 GHz†, 4 VCPUs 4 GB RAM 8 GB Virtual disk 1 Gbps Ethernet	Intel® Xeon E5-2680 v2 2.8 GHz, 8 VCPUs 15 GB RAM 160 GB SSD Amazon Enhanced Network

†We limit the CPU to 2.7 GHz and disable Turbo boost.

Figure 3: VM and HW specs at offloading sites

Smartphone (Samsung Galaxy Nexus)	Netbook (Dell Latitude 2120)
ARM® Cortex-A9 1.2 GHz, 2 cores 1 GB RAM 32 GB Flash 802.11a/b/g/n WiFi	Intel® Atom™ N550 1.5 GHz, 2 cores 2 GB RAM 250 GB HDD 802.11a/g/n WiFi

Figure 4: HW configuration of mobile devices

connection to EC2-East is much faster (8 ms) and is atypical. Because our organization is a major Amazon AWS customer, we have preferential routing in the Internet backbone to EC2-East. Such routing would not be available to a mobile device in the field. Thus, in interpreting our results EC2-East should be considered the best possible case, while EC2-West should be regarded as typical. Unless qualified, the term "cloud" refers to EC2-West in the rest of this paper.

Figure 5 illustrates the networking used in our experiments. There are five configurations, described below.

No Offload: The server component runs locally on the mobile device, thereby avoiding all network transmission.

Cloud-WiFi: The mobile device uses 802.11n to connect to a private WiFi access point that is connected to an enterprise network via Ethernet, and thence via the Internet to an Amazon AWS site.

Cloudlet-WiFi: This is similar to the Cloud-WiFi configuration, except that network traffic only needs to go as far as to the cloudlet, which is on the same Ethernet segment as the WiFi access point.

Cloudlet-LTE: Under a license for experimental use from the FCC, we have set up an in-lab 4G LTE network. This in-lab cellular network uses a Nokia eNodeB whose transmission strength is attenuated to 10 mW. Network traffic via the eNodeB is directed through a Nokia RACS gateway and local Ethernet segment to the cloudlet.

Cloud-LTE: Due to limitations in the network setup between our in-lab cellular network and the Internet, our lab cell cannot achieve the typical performance for the Cloud-LTE option. So we use cellular data service on the commercial T-Mobile 4G LTE network to reach the Internet, and thence an Amazon AWS site.

Because of the experimental frequency bands used in our in-lab LTE network, only a few phones such as the Google Nexus 6 can connect to it. Unfortunately, `COMET` cannot run on this phone. So for the cloudlet-LTE experiments, we use USB and WiFi tethering on Google Nexus 6 to connect the netbook and `COMET`-capable smartphone respectively. For consistency, we use tethering with Cloud-LTE as well.

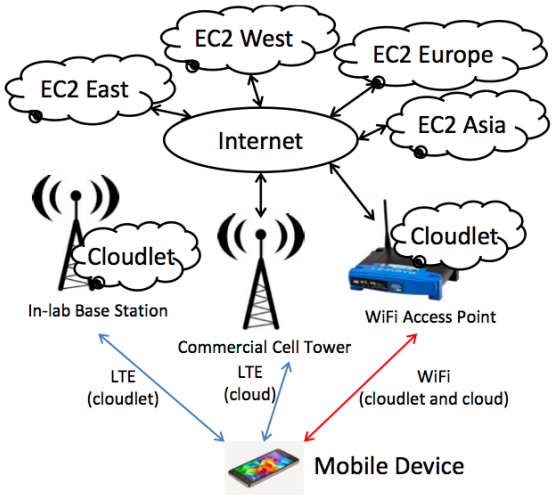


Figure 5: Network setup for the experiments

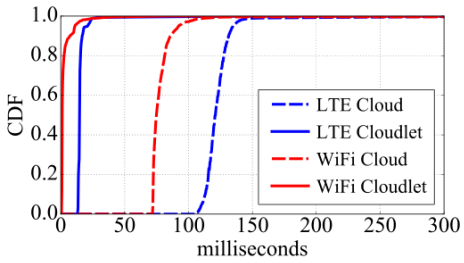


Figure 6: CDF of pinging RTTs

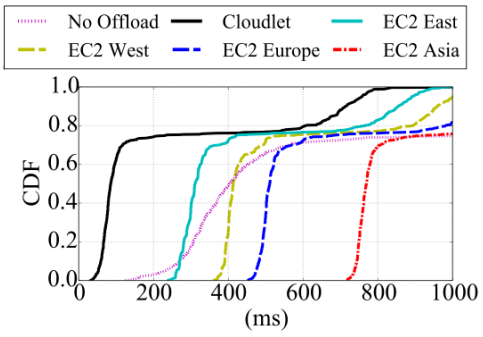
Figure 6 shows the RTTs of all the above network configurations except No Offload, using the Linux `ping` command with 1 ms pinging interval. The ping RTTs represent the minimum baseline latencies possible for our network configurations.

3. Experimental Results

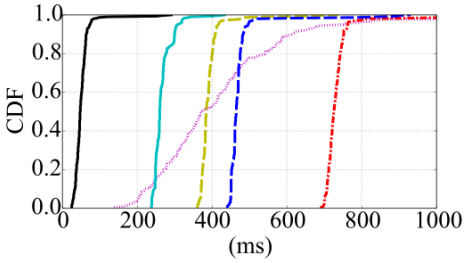
3.1 WiFi Offloading Performance

Figure 7 compares response times when offloading prepartitioned applications to Amazon AWS sites, or to a cloudlet. Also shown is the application response time without offload. The graphs plot cumulative distribution functions (CDFs) of response times measured over three runs.

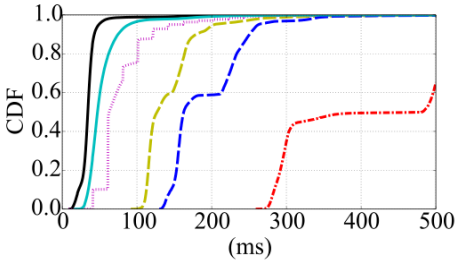
Response times clearly degrade with increasing latency to the offload site. For all three applications, the CDF curves shift right due to higher network latency and lower effective bandwidth to the farther offload sites. The CDFs for more remote sites also rise less sharply, indicating greater variability of response times, due to greater uncertainty of WAN network conditions. (These effects are somewhat masked by the large inherent variation in performance for the `FACE` application.) As a result, *No Offload* often provides faster response times than offloading to distant clouds. The cloudlet consistently provides the best response times.



(a) FACE (300 images per run)

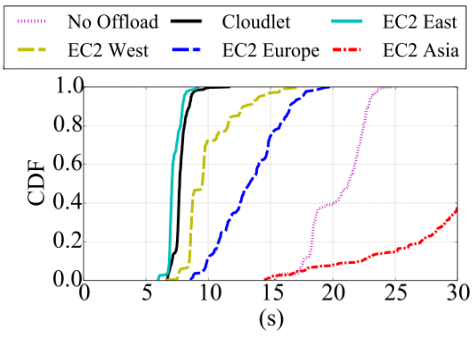


(b) MAR (100 images per run)

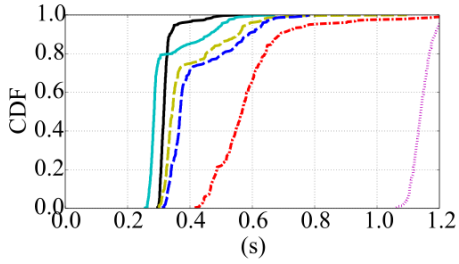


(c) FLUID (10 min runs)

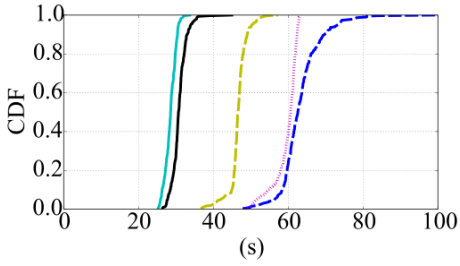
Figure 7: Response times for prepartitioned apps (WiFi)



(a) Linpack



(b) CPU Benchmark



(c) PI Benchmark

Figure 8: Execution times for COMET apps (WiFi)

Figure 8 presents the results for COMET-based applications. These, too, show that response time is affected by offload location. EC2-West, EC2-Europe, and EC2-Asia are clearly worse than EC2-East. For PI Benchmark, EC2-Asia is too far to the right to be visible. However, these longer running applications (seconds to tens of seconds, rather than tens to hundreds of milliseconds) change the relative performance of EC2-East and the cloudlet. As mentioned in Section 2.2, latency from our testbed to EC2-East is unusually good and not much higher than WiFi latency. As a result, the greater computational power of the cloud instances over the cloudlet now becomes significant, so EC2-East is the fastest.

The results also show the impact of application characteristics. CPU Benchmark transfers very little data, so the various EC2 curves look very similar, mainly shifted by the difference in RTT to the different sites. Linpack transfers significantly more data, and the differences in response times between EC2 sites are much more apparent. PI Benchmark transfers a similar amount of data in total, but in multiple transfers, as the processing thread is transferred back and forth between the mobile device and backend. As a result, for PI Benchmark, the network effects of distant clouds are magnified. We delve further into the effects of multiple migrations in Section 3.3.

3.2 WiFi Offloading and Energy

In addition to improving performance, offloading computation can reduce the energy consumption on mobile devices. We rerun our WiFi offload scenarios while measuring the power consumption of the mobile devices. For the netbook, we remove the battery and use a WattsUp power meter [27] to log the total power draw at the AC power cord. On the smartphone, we interpose a Monsoon Power Monitor [18] meter at the battery contacts, logging voltage and current draw during execution of our benchmarks.

The average energy consumed per request or per run are shown in Figure 9, along with standard deviations in parentheses. Offload to any site greatly reduces power (and therefore energy consumed) on the mobile devices, since they no longer perform heavy computations. The choice of offload site does not affect power, which is about the same for

	Offload	None	Cloudlet	East	West	Europe	Asia
Face†	(J/query)	12.4 (0.5)	2.6 (0.3)	4.4 (0.0)	6.1 (0.2)	9.2 (4.1)	9.2 (0.2)
Fluid†	(J/frame)	0.8 (0.0)	0.3 (0.0)	0.3 (0.0)	0.9 (0.0)	1.0 (0.0)	2.2 (0.1)
MAR†	(J/query)	5.4 (0.1)	0.6 (0.1)	3.0 (0.8)	4.3 (0.1)	5.1 (0.1)	7.9 (0.1)
Linpack	(J/run)	40.3 (2.6)	13.0 (0.7)	13.3 (2.3)	16.9 (1.8)	18.2 (1.9)	38.1 (4.1)
CPU	(J/run)	9.6 (1.4)	5.7 (0.3)	5.9 (0.3)	5.8 (0.3)	5.9 (0.2)	6.0 (0.2)
PI	(J/run)	129.7 (2.9)	53.9 (2.1)	57.6 (1.8)	107.6 (8.6)	162.8 (18.0)	203.4 (16.7)

Numbers in parentheses are standard deviations from three runs. †The display is turned off during energy measurement.

Figure 9: Energy consumption on mobile devices

cloudlet and cloud cases. However, execution duration does vary, so the actual energy consumed per query/frame/run is greatly affected by offload site. In all cases, cloudlet offload results in the least energy consumed per execution, while further offload sites incur increasing energy costs. In fact, in many cases, offloading to EC2-Asia or EC2-Europe results in higher energy costs than not offloading would.

3.3 Effects of Interactivity

Although COMET is effective at offloading the CPU-intensive portions of unmodified Android applications, any interaction with device hardware (e.g., sensor I/O, screen update, or file operation), will require application threads to migrate back to the mobile device. This can cause many thread migrations over the course of the program execution, magnifying the effects of network latency and bandwidth.

To study this effect of interactivity further, we create a custom CPU-intensive application that factors large integers. For purposes of offloading, COMET treats file I/O as interaction, similar to display updates. When an application thread attempts any interaction, COMET migrates it back to the mobile device before allowing the operation to proceed. Our app saves intermediate results to a file after a configurable number of iterations, allowing us to vary how often migrations are triggered with COMET. Figure 10 shows how the app performance changes as the interactivity level (number of file operations) increases. The execution time without offloading remains nearly constant because the application is CPU bound on the mobile device, even with a few hundred file operations. However, cloud offloading performance is highly affected, and becomes slower than just running on the mobile device once we exceed 75 I/O operations during the execution. In contrast, offloading to a cloudlet is much less sensitive to the interactivity level. Until 240 file writes, it performs better than the no-offloading case. These results show that cloudlets can greatly benefit frameworks like COMET by muting the effects of application interactivity. Independent of our work, *Tango* [10] corroborates that interactivity of an application critically affects code offloading. This work uses deterministic replay to improve offloading performance. In contrast, we advocate using cloudlets to completely avoid long response times.

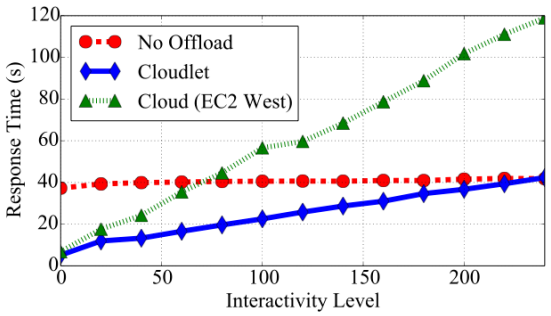
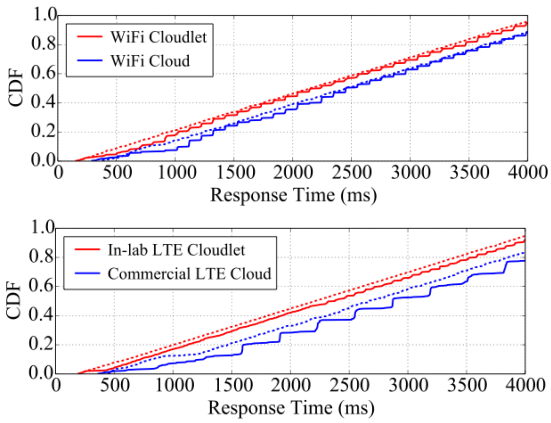


Figure 10: Effect of Interactivity on COMET (WiFi)



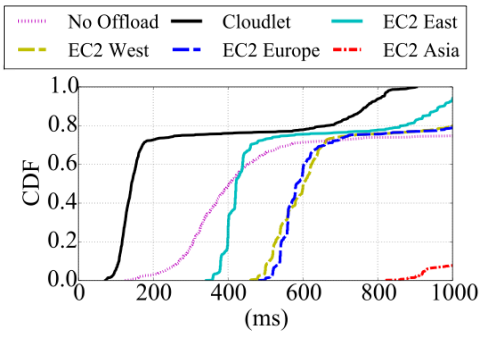
Dotted lines are with periodic pings (1 ms intervals).

Figure 11: Effect of link power management on echo server response times with smartphone

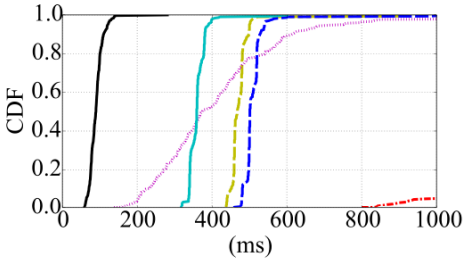
3.4 Effects of Radio Link Management

Our initial experiments with a smartphone on LTE showed that network latencies tend to jump between discrete values. Figure 11 shows the CDF of the observed response times over LTE and WiFi from a simple echo server that gradually increases the delay before sending a reply. We would expect a straight line, indicating uniform distribution of response times, but the results show a distinct stair-step pattern. As steps are visible in both LTE and WiFi, we believe this stair-step pattern is due to aggressive power down of the radio by the mobile device. This effect is not due to CPU scheduling, and is not present in WiFi results from non-phone devices tested. For LTE, we observe two different steps sizes, which start to occur at different times. The latter is likely due to greater aggressiveness of the commercial LTE network configuration in reclaiming idle resources. We believe the different step size is due to the use of discontinuous reception mode (DRX) in LTE [2]. This mechanism uses short and long sleep cycles to let the mobile device power down radio circuitry. The use of DRX and sleep durations are determined by the LTE network.

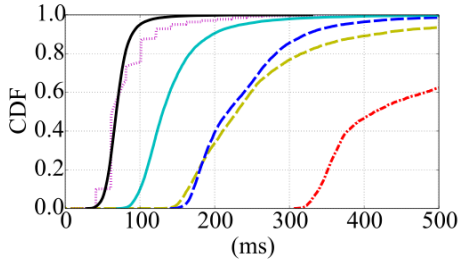
Our experiments also show that if the mobile device performs frequent data transmissions (e.g., periodic background



(a) FACE (300 images per run)



(b) MAR (100 images per run)



(c) FLUID (10 min runs)

Figure 12: Response times for prepartitioned apps (LTE)

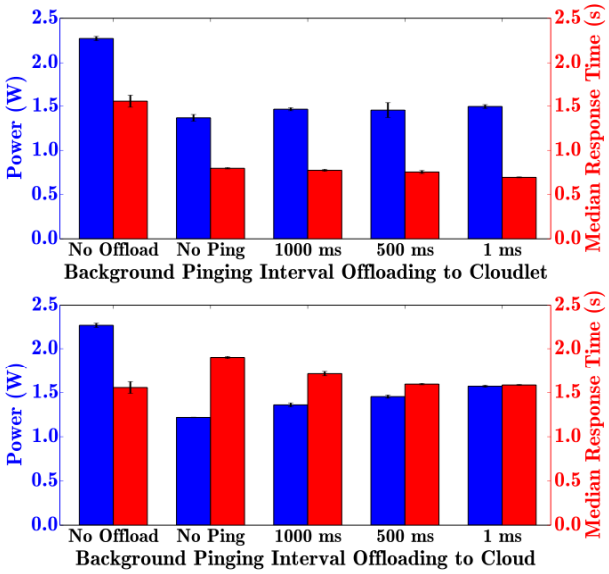
pings), these effects completely disappear. To avoid the confounding effects of latency discretization, all of our LTE experiments are run with background pings at 1 ms intervals.

3.5 LTE Offloading Performance

Figure 12 shows the response times for the three prepartitioned applications when using LTE. Overall, these results are quite similar to those from the WiFi experiments. As the LTE latencies are slightly higher, the curves are shifted to the right compared to the WiFi results. Additionally, the commercial LTE network does not have as good connectivity to the EC2-East data center as our WiFi setup does. So, although the EC2-East response times are still the best among the cloud scenarios, they are not quite as low as in our WiFi experiments. In all cases, offloading to the cloudlet provides the best response times.

3.6 LTE Energy Tradeoff

In the previous LTE experiments, we use background pinging at 1 ms intervals to overcome discretized latency effects from Section 3.4 and minimize response times. However, this approach can negatively impact mobile energy consumption. Here, we quantify this tradeoff between response time and energy consumption by varying the ping intervals.



The error bars show standard deviation across three runs. Each run uses 250 images.

Figure 13: Response time and power consumption for Tesseract-OCR (LTE)

Since we need a tethering setup to run the selected applications in our in-lab LTE network, energy measurement becomes difficult. We therefore use a different application (Tesseract-OCR [9]) that can run directly on the Google Nexus 6. We measure energy consumption using the battery voltage and current reported by the OS[1], since the Nexus 6 smartphones do not have removable batteries.

Figure 13 shows the response time and power dissipation of Tesseract-OCR, comparing local execution (leftmost bars) with offloading to cloudlet (top plot) and offloading to cloud (Amazon EC2-West, bottom plot). Offloading, even without background pings, clearly reduces power dissipation on the mobile device. When offloading to the cloud, this comes with a response time penalty. In contrast, on the cloudlet, both the latency and the energy consumption are reduced. Adding background pinging at increasing frequencies can improve response time at the expense of increased power. When offloading to the cloud, pinging every 1 ms (rightmost two bars) reduces the response time by 17% but increases the power by 29% compared to the no pinging case (second pair of bars). Since our in-lab LTE network does not instruct the mobile device to save power as aggressively as the commercial network does, the effect of adding

[1] In `/sys/class/power_supply/battery/current_now` and `/sys/class/power_supply/battery/voltage_now`

background pinging is not as significant when offloading to the cloudlet, though the overall trend is still visible. When consistent low response times are paramount, adding background pinging can be a valuable tool to reduce the latency effects of link management in LTE.

4. Related Work

This work quantifies the relative merits of cloudlet offload versus cloud offload. It takes an application-centric view, recognizing that a mobile user craves crisp interaction and long battery life. Achieving these attributes on small, lightweight mobile hardware for compute-intensive and memory-intensive applications such as speech recognition and face recognition is only possible through offloading. This work spans diverse applications, considers static and dynamic partitioning strategies, examines both WiFi and 4G LTE wireless networks, explores the role of application interactivity, and uncovers some idiosyncrasies of 4G LTE networks.

To the best of our knowledge, no other work provides such an extensive and detailed investigation of cloud offload versus cloudlet offload. Closest in spirit to this work are the measurements reported in 2013 by Ha et al. [13]. That work only examined prepartitioned applications and WiFi networks; it did not consider dynamically partitioned applications or 4G LTE networks.

Of broader relevance is the long history of work on cyber foraging. The foundational work of Noble et al. [19] and Flinn et al. [8] have already been mentioned in Section 1. Detailed accounts of the evolution of cyber foraging are provided in the survey by Flinn [7] and the retrospective by Satyanarayanan [21].

Related to our study of offloading over 4G LTE networks is the extensive 2012 study of wide-area wireless networks by Huang et al. [14]. That work examines a wide range of networking technologies, but only considers today's web applications. In contrast, our work considers applications that are representative of future workloads for edge computing. A direct point of comparison is the observed RTT over commercial 4G LTE networks. Huang et al. report mean values around 70 milliseconds, which is consistent with our measurements considering the difference in experimental setup. These values are in contrast to the roughly 15 millisecond RTT observed on our experimental 4G LTE network, suggesting that much of the latency is incurred in the evolved packet core (EPC) of a commercial 4G LTE network rather than in its RF segment.

5. Conclusion

Our study has shown that the choice of offloading site is important for both prepartitioned applications and for dynamic offloading frameworks such as `COMET`. In comparison to the average cloud (EC2-west), we show that edge computing can improve response time and energy consumption significantly for mobile devices. These advantages are not limited to WiFi networks. Even when offloading over LTE, edge computing continues to provide superior results.

Our results also show that offloading computation blindly to the cloud can be a losing strategy. Offloading to a distant cloud can result in even lower performance and higher energy costs than running locally on the mobile device. For highly interactive applications, even offloading to nearby clouds can be detrimental to performance. Such applications demonstrate most clearly that edge computing is necessary to achieve performance and energy improvement through offloading.

There is increasing commercial interest in deeply immersive applications such as mobile augmented reality and cognitive assistance. For such interactive and compute-intensive applications, our results show that the industrial efforts towards building edge computing infrastructure are not misplaced. Rather such infrastructures will be key enablers of this new genre of applications.

6. Acknowledgements

This work has been supported by the National Science Foundation (NSF) under grant number CNS-1518865. Additional support was provided by Intel, Vodafone, Google, Crown Castle, and the Conklin Kistler family fund. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and should not be attributed to Carnegie Mellon University or the funding sources.

7. REFERENCES

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, Helsinki, Finland, 2012.

[2] C. S. Bontu and E. Illidge. DRX mechanism for power saving in LTE. *IEEE Communications Magazine*, 47(6):48–55, 2009.

[3] G. Bradski et al. The opencv library. *Doctor Dobbs Journal*, 25(11):120–126, 2000.

[4] G. Brown. Converging Telecom & IT in the LTE RAN. White Paper, Heavy Reading, February 2013.

[5] Z. Chen, L. Jiang, W. Hu, K. Ha, B. Amos, P. Pillai, A. Hauptmann, and M. Satyanarayanan. Early implementation experience with wearable cognitive assistance applications. In *Proceedings of the 2015 workshop on Wearable Systems and Applications*, pages 33–38. ACM, 2015.

[6] P. Cokulov. Linpack - Android Apps on Google Play. https://play.google.com/store/apps/details?id=rs.pedjaapps.Linpack, 2015.

[7] J. Flinn. *Cyber Foraging: Bridging Mobile and Cloud Computing via Opportunistic Offload*. Morgan & Claypool Publishers, 2012.

[8] J. Flinn and M. Satyanarayanan. Energy-aware Adaptation for Mobile Applications. In *Proceedings of the 17th ACM Symposium on Operating Systems and Principles*, Kiawah Island, SC, December 1999.

[9] Google. Tesseract Open Source OCR Engine. https://github.com/tesseract-ocr.

[10] M. S. Gordon, D. K. Hong, P. M. Chen, J. Flinn, S. Mahlke, and Z. M. Mao. Accelerating mobile applications through flip-flop replication. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 137–150. ACM, 2015.

[11] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen. COMET: Code Offload by Migrating Execution Transparently. In *Proceedings of the 10th USENIX Symposium on Operating System Design and Implementation*, Hollywood, CA, October 2012.

[12] K. Greene. AOL Flips on 'Game Changer' Micro Data Center. http://blog.aol.com/2012/07/11/ aol-flips-on-game-changer-micro-data-center/, July 2012.

[13] K. Ha, P. Pillai, G. Lewis, S. Simanta, S. Clinch, N. Davies, and M. Satyanarayanan. The Impact of Mobile Multimedia Applications on Data Center Consolidation. In *Proceedings of the IEEE International Conference on Cloud Engineering*, San Francisco, CA, March 2013.

[14] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. A Close Examination of Performance and Power Characteristics of 4G LTE Networks. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, Lake District, UK, 2012.

[15] Intel. Intel Integrated Performance Primitives. https://software.intel.com/en-us/intel-ipp.

[16] A. Li, X. Yang, S. Kandula, and M. Zhang. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th annual conference on Internet measurement*, pages 1–14. ACM, 2010.

[17] S. D. Markovic. PI Benchmark - Android Apps on Google Play. https://play.google.com/store/apps/details?id=rs.in.luka.android.pi, 2015.

[18] Monsoon Solutions. Power Monitor. http://www.msoon.com/LabEquipment/PowerMonitor/.

[19] B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker. Agile Application-Aware Adaptation for Mobility. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, Saint-Malo, France, October 1997.

[20] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, 8(4), 2001.

[21] M. Satyanarayanan. A Brief History of Cloud Offload: A Personal Journey from Odyssey Through Cyber Foraging to Cloudlets. *GetMobile: Mobile Computing and Communication*, 18(4), January 2015.

[22] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4), October-December 2009.

[23] B. Solenthaler and R. Pajarola. Predictive-corrective incompressible SPH. *ACM Trans. Graph.*, 28(3):40:1–40:6, July 2009.

[24] G. Takacs, M. E. Choubassi, Y. Wu, and I. Kozintsev. 3D mobile augmented reality in urban scenes. In *Proceedings of IEEE International Conference on Multimedia and Expo*, Barcelona, Spain, July 2011.

[25] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[26] Unstable Apps. CPU Benchmark - Android Apps on Google Play. https://play.google.com/store/apps/details?id=com.unstableapps.cpubenchmark, 2015.

[27] WattsUp. .NET Power Meter. http://wattsupmeters.com/.