FemtoClouds: Leveraging Mobile Devices to Provide Cloud Service at the Edge

Karim Habak*, Mostafa Ammar*, Khaled A. Harras†, Ellen Zegura*
*School of Computer Science, College of Computing, Georgia Institute of Technology
†Computer Science Department, School of Computer Science, Carnegie Mellon University
Emails: *{karim.habak, ammar, ewz}@cc.gatech.edu, †kharras@cs.cmu.edu

***Abstract*—Mobile devices are becoming increasingly capable computing platforms with significant processor power and memory. However, mobile compute capabilities are often underutilized. In this paper we consider how a collection of co-located devices can be orchestrated to provide a cloud service at the edge. Scenarios with co-located devices include, but are not limited to, passengers with mobile devices using public transit services, students in classrooms and groups of people sitting in a coffee shop. To this end, we propose the femtocloud system which provides a dynamic, self-configuring and multi-device mobile cloud out of a cluster of mobile devices. We present the femtocloud system architecture designed to enable multiple mobile devices to be configured into a coordinated cloud computing service despite churn in mobile device participation. We develop a prototype of our femtocloud system and use it in addition to simulations to evaluate the performance of the system showing its efficiency and ability to leverage the available devices' compute capacity. We contribute to a line of research on small, local and possibly private clouds.**

I. Introduction

Since the 2002 paper by Balan et al. making a case for mobile devices to cyberforage by finding surrogate (i.e., helper) servers in the environment [1] the research community has explored various forms of interaction between mobile devices and fixed, higher capacity infrastructure, including the cloud. The motivation for this exploration has been and remains as articulated by Satyanarayanan[14], namely that mobile devices are resource constrained in comparison with servers, and that users desire high performance applications regardless of the device used to experience the application. By offloading some computation to more powerful servers, mobile devices can offer a user experience beyond what local capabilities can support. Further, offloading may allow mobile devices to save power and extend time between charges.

In addition to questions of performance speedup, energy savings and cost, the key questions for an offloading system design are: where is the higher performance capacity, who provides it, and how does it fit into a larger computing ecosystem? In traditional cloud computing, the higher performance capacity is located in data centers reached via the Internet and provided by companies that charge for transient server use. Traditional cloud computing can offer essentially unlimited compute capacity, but at the price of latency and bandwidth limitations between the mobile device and the servers in large data centers. In response to these limitations, the Cloudlet system moves computation closer to mobile devices, creating a two-tier architecture where a mobile device can offload to a nearby, less capable server, at low latency and high bandwidth, rather than (or as a complement to) offloading to the cloud [15]. In the cloudlet vision, these nearby servers would be located in public and commercial spaces where people congregate, such as coffee shops and airport waiting areas [15]. One could imagine a third-party provider owning and operating these cloudlets for profit.

We make two observations about the target environment for cloudlets that motivate our work. First, while the gap between truly mobile devices (handhelds, wearable) and high capacity servers remains [5], mobile devices have grown increasingly powerful, especially when laptops are included. Second, from an architectural perspective, it is possible to refactor the cloudlet into a *controller* – responsible for receiving tasks, scheduling their computation, and returning results – and a *compute cluster* responsible for performing the computation. Putting these two observations together motivates our research questions: when and how can user devices be combined with a controller to form a deconstructed cloudlet or “femtocloud”? And why would that be preferred over a traditional cloudlet?

As a start, configuring a useful femtocloud requires a collection of nearby devices with sufficient idle capacity and sufficient stability to form a compute cluster. It is natural to consider settings where people congregate for time periods, often with personal devices, such as coffee shops, classrooms or theaters, and public transportation. These example settings share additional properties that help make femtoclouds feasible:

- There is a natural owner of the setting who may have a business interest in providing (or contracting for) the controller that makes the femtocloud work, whether a coffee shop owner, a university, a theater owner, or a public transport provider.
- Each setting has semantics that suggest forms of stability and, importantly, predictability in the duration of time that a given device is available for use in the femtocloud. A classroom has pre-determined time periods of use – during a scheduled class – and predetermined times when the occupancy will experience most turnover. Public transportation offers only fixed chances for occupancy change, based on bus or train stops. A coffee shop is more complicated from a stability and predictability standpoint; for now, we simply observe that coffee shop patrons fall

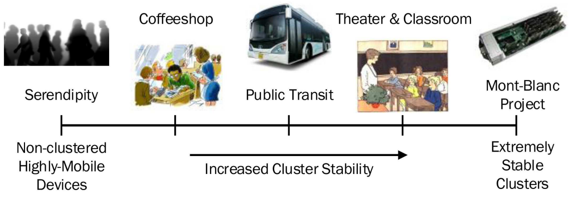

Coffeeshop
Theater & Classroom
Serendipity
Public Transit
Mont-Blanc Project
Non-clustered Highly-Mobile Devices
Increased Cluster Stability
Extremely Stable Clusters

Fig. 1. Mobile cluster stability spectrum.

into at least two classes – those who stand in line, make a purchase, and leave immediately, and those who linger.

- There is a potential to build trust based on in-person, social relationships. Coffee shop patrons, students at a university, and public transport riders are typically repeat customers with a relationship of some form to the owner of the setting. Repeat participation also provides the potential to learn about devices and device owners in ways that can optimize femtocloud usage.
- There is a natural form of payment to those who participate, associated with the setting, such as coffee shop credit, university currency credit, or public transportation credit. While other forms of compensation for device use are certainly possible, these options connect the place where the device is used to the compensation in ways that may be attractive to device owners and the setting owner.

The second property turns out to be critical for scoping our work and for operating a femtocloud. We can situate femtoclouds relative to other approaches to use mobile devices to provide a compute service, by examining the stability and predictability along a spectrum as shown in Figure 1. At one end of the spectrum are extremely stable and deliberately configured clusters such as proposed in the Mont-Blanc project (www.montblanc-project.eu) where, motivated by energy considerations, a large number of mobile CPUs are configured in a single chassis. Our settings of interest – a coffee shop, public transit and theater/classroom – fall in the middle of the spectrum. At the other end of the spectrum are highly mobile and unpredictable devices that are used opportunistically as they are encountered (e.g., over time [17], [18], [11]).

It is natural to ask what advantages a femtocloud might have in comparison to a cloudlet. We identify three possibilities. First, by tapping the compute resources in a setting of mobile devices who are offloading computation, femtoclouds have a natural scaling property not enjoyed by a cloudlet, which must be provisioned ahead of time. The more devices in the setting, the more potential for those devices to have tasks they want to offload, but also the more potential exists for idle resources to use in serving offloaded tasks. Second, a femtocloud confines the controlled, owned infrastructure to just the controller, producing a capability much closer to a co-operative or community-based computing service that does not rely heavily on a corporate service. Finally, and philosophically, femtoclouds push the cyberforaging vision a step further to rely on infrastructure only for control and coordination, but not for compute cycles.

In this paper we develop and evaluate the femtocloud system. We begin in Section II with the architecture, identifying functionality to be realized in the controller and in the mobile devices, and information to communicate within and between the two. We identify a critical and obvious problem that must be solved at the controller, namely the scheduling of tasks onto mobile devices where the transmission of data and receipt of results all happens over a shared wireless channel. We formalize the scheduling problem and then develop several algorithms in Section III. We evaluate the system in Section IV using simulations, including those driven by measurements of device dynamics in different settings. We also describe and report briefly on a prototype built on Android. We end the paper with Related Work and Conclusions.

II. The FemtoCloud System

The femtocloud computing service executes a variety of tasks that arrive at the control device. The femtocloud client service, running on the mobile devices, estimates the computational capability of the mobile device, and uses it along with user input to determine the computational capacity available for sharing. This client leverages device sensors, user input, and utilization history, to build and maintain a user profile. Afterwards, the service shares the available information with the control device, which is then responsible for estimating the user presence time and configuring the participating mobile devices as a cloud offering compute as a service.

In this section, we present the details of the femtocloud system. We start with listing our assumptions followed by the detailed description of our architecture depicted in Figure 2. Afterwards, we present the implementation details of our prototype.

A. Assumptions

We assume that some users will have the femtocloud client service installed on their mobile devices, and that they are willing to share a portion of their computational capabilities as a result of different incentives ranging from their willingness to share resources (as in SETI or BOINC) to direct financial gains. We acknowledge that such incentive mechanism is essential specially for users with battery operated devices. We assume that a femtocloud controller is responsible for deciding which mobile devices will be added to the compute cluster in the current environment.

We assume a general task arrival model where tasks can arrive individually or in batches following any task arrival distribution. Each of these tasks is a *compute intensive* tasks that has its own computation requirements, input data size, and output data size. We assume that a task assigned to a mobile device needs to be completed and the results returned to the control device before the mobile device leaves the cluster. Otherwise, the task is aborted and may need to be re-assigned and restarted. Based on these task parameters as well as the availability of mobile devices, the controller builds a task execution schedule and assigns each task to a mobile device to optimize the metric of interest.

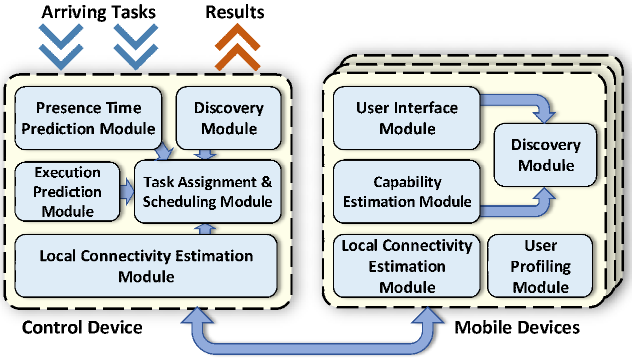

Arriving Tasks
Results
Presence Time Prediction Module
Discovery Module
Execution Prediction Module
Task Assignment & Scheduling Module
Local Connectivity Estimation Module
Control Device
User Interface Module
Discovery Module
Capability Estimation Module
Local Connectivity Estimation Module
User Profiling Module
Mobile Devices

Fig. 2. The femtocloud system architecture

B. System Architecture

The mobile device functions are performed in the following modules:

User Interface Module: This module obtains user preferences, resource sharing policies and personal profile sharing policy. For instance, the user can configure this module to share up to certain percentage of his mobile device capabilities, or contribute to femtocloud only if the battery level is above certain threshold. They also define policies that dictates whether they are willing to join a femtocloud or not. These policies are defined by many factors such as available battery level, time, environment type, etc.

Capability Estimation Module: This module estimates the computational capabilities of the mobile device including the number of cores in the device and the available computational capacity. Such computation capacity varies based on the system load and whether the device running a power savings mode. The computational capacity estimate is shared with the control device. Since the estimate may change over time, this module periodically sends updated estimates to the control device.

User Profiling Module: This module gather data about the user preferences and behavior in different scenarios to be used for determining his presence time while joining femtocloud. This module opportunistically mines the gathered data and build a profile. This module only share the profile with the controller in user accepted granularity to maintain user privacy.

The control device functions are performed by the following modules:

Execution Prediction Module: In order to efficiently distribute tasks across different processing nodes, the controller should know the execution load introduced by each of these tasks. To achieve this goal, we rely on the original task source to provide the controller with this information. However, if the source does not provide such information, the control device carries the responsibility of connecting to an execution estimation service to acquire it. Such estimation may be done using the Mantis system [9].

Presence Time Prediction Module: This module is responsible for predicting the presence time for femtocloud users. It gathers environment specific data to build a generic user profile based on the collective behaviors of the users. This profile is used to estimate the presence time for new users as well as updating the estimates over time. It also uses specific user profile, if shared, along with this generic profile to determines his presence time.

Task Assignment and Scheduling Module: This module uses the information acquired by the previous modules to iteratively assign tasks to their executing devices.

The control device collaborates with the mobile devices in the cluster to perform functions implemented in the following modules which are instantiated in both types of device.

Local Connectivity Estimation Module: This module estimates the available bandwidth between the control device and each computing mobile device. Since these devices are directly connected and relatively static in most of the scenarios, many techniques can be used to estimate the available bandwidth. Our approach is to use the wireless signal strength to get the initial estimate of the bandwidth and then monitor the actual achievable bandwidth while assigning tasks and/or gathering results to update such estimate.

Discovery Module: This module discovers the available mobile devices that have the femtocloud client service installed. Once a mobile device becomes ready to join a cluster, it sends a registration packet to the control device. This registration packet can includes an initial estimate of the compute capacity based previous contribution to the femtocloud system in similar context and user profile information to be used for determining his presence time. We also use periodic heartbeats to keep track of the devices in the cluster and gather more updated information about their shared computational capacity.

C. Implementation

To assess the feasibility of femtocloud and evaluate it, we implement a femtocloud prototype in Android.

We implement the control device to hold the responsibility of providing an interface to the task originators and to manage the mobile devices inside the cloud. The interface to the task originators enables them to send the code for the desired computation coupled with their input data and to receive the results once they become available. The control device works in collaboration with the femtocloud client service installed in the mobile devices to acquire information about the device characteristics and user profiles. The service uses such information to assign task to devices according to a heuristic which will be described in the next section. To minimize the communication overhead and enhance performance we first use persistent TCP connections between each mobile device and the control device to avoid the delays introduced by the protocol's handshaking and slow start mechanisms. We also allow the control device to act as a WiFi hotspot allowing the mobile devices to connect to it using infrastructure mode.

Note that there is no contention for the communication channel between the control device and the mobile devices in the cluster due to our scheduling technique. For communication from the mobile devices to the control device there will be two types: 1) short notifications and alerts that are allowed at any time and may contend for the channel, and 2) possibly longer communication needed to return computation results to the controller which are *scheduled* by the control device.

We implement the femtocloud client service as an Android application that allows the user to enter preferences and resource sharing policies. Once a user accepts to share a portion of the mobile device's resources and select the granularity of sharing his profile with the controller, it connects to the WiFi network offered by the controller. Upon successful connection, this service holds the responsibility of estimating the mobile device capabilities and sharing them with the controlling device. In addition, it works in collaboration with the control device to estimate the available bandwidth between the mobile device and the control device as well as the user presence time. More importantly, it carries the responsibility of executing tasks assigned to it by the controller. Upon completing the execution of a task, the client service stores the results, notifies the control device regarding the availability of such results, and starts executing other assigned tasks, if any. Finally, once it receives a request for the results from the controller, it sends the available results to the controller, deletes them and erases any stored state information about the task.

III. FemtoCloud Scheduling Problem

The scheduling algorithm that runs at the controller is critical to the performance of the system. The scheduler must assign tasks to available devices to maximize the metric of interest, while managing device churn. The task assignment problem differs from standard parallel task assignment because sending and receiving tasks takes place over a shared wireless channel and because we assume that if a device departs prior to completing and delivering its task result, the task must be reassigned and restarted from the beginning. These two constraints place a priority on getting tasks assigned quickly, executed well within estimates of device persistence, and results returned quickly to the controller. While other metrics are possible, we focus on maximizing the "useful computation", defined as total computation completed by the system.

We begin by formulating the problem as an optimization problem, assuming perfect knowledge of device capabilities (computation and bandwidth) and departure time. We then describe a greedy heuristic based on insights gained by solving the optimization problem on small instances.

A. Scheduling as Optimization

Table I summarizes the system parameters and notation used in the optimization. We assume the scheduler must distribute a batch of n tasks across the available mobile devices and gather their results. We assume that our cluster consists of m mobile devices with users willing to share their computation capabilities. Let $\mathcal{C}_k$ denote the shared computation capability of the k^{th} mobile device, $\mathcal{B}_k$ denote the available communication bandwidth between this device and the controller, and $\mathcal{T}_k^d$ denote the departure time of this device. For each of the n tasks, $\mathcal{E}_i$ denotes the execution load introduced by task i, I_i denotes the size of the code and its inputs, and R_i denotes the size of results of the same task.

Our goal is to determine a complete task-execution schedule. For each task, the scheduler determines which device to assign the task to, when to send the task to the device, and when to schedule the return of the result. The overall objective of the task-execution scheduler is to maximize the overall cluster's useful computations:

TABLE I
LIST OF SYMBOLS USED

Symbol	Description
$\mathcal{B}_k$	The available bandwidth at the k^{th} device
$\mathcal{C}_k$	The shared processing capacity of the k^{th} device
$\mathcal{T}_k^d$	The departure time of the k^{th} device
$\mathcal{E}_i$	The execution load of the i^{th} task
I_i	The size of the transferable input data and executable code of the i^{th} task
R_i	The size of the results of the i^{th} task
x_{ik}	Equals 1 if the i^{th} task is assigned to the k^{th} device and equals 0 otherwise
n	Number of tasks waiting for assignment
m	Number of devices in the cluster

$$\text{Maximize } \mathcal{C} = \sum_i \mathcal{E}_i \sum_k x_{ik} \tag{1}$$

where $\mathcal{C}$ is the completed computational load and $\mathcal{E}_i$ is the execution load introduced by the i^{th} task.

The decision variables are: (1) $x_{ik}, \forall i, k$ where $x_{i,k}$ equals 1 if task i is assigned to device k and equals 0 otherwise. (2) The times of assigning a task to a processor, executing it, and sending its results to the controller. Therefore, solving this optimization produces not only a task assignment table but also a complete schedule for task transmission, execution and results transmission.

The following constraints must be satisfied (we omit the mathematical formulation of these constraints due to space constraints):

1) Each task should be assigned to at most one mobile device.
2) To ensure the schedule correctness, each task should be assigned enough time to completely send its code and input data before beginning execution. It should also be assigned enough time to completely execute at the mobile device as well as sufficient time send its results to the controller.
3) To ensure the usefulness of the task computations, all the results of the tasks assigned to a mobile device should be sent and stored at the controller prior to the departure of the device.
4) To maximize the task results availability, tasks that are assigned to the same processor/device should not be concurrently executed.
5) To avoid any delay introduced by wireless media contention, the scheduler should not allow simultaneous use of the wireless channel to send different tasks to their executing nodes and to receive finished tasks results.

Generally, this task assignment problem is a mixed 0-1 integer programming problem that can be shown to be NP-Complete. However, this problem definition guides us towards developing heuristics for task assignment and gathering of results.

B. *Heuristics*

Task Assignment Heuristics: To provide an efficient solution for our task assignment problem, we adopt an iterative greedy approach to assigning tasks to mobile devices. Our approach is based on three key ideas: (1) To maximize the efficiency of utilizing the communication channel, tasks with higher computational requirement per unit data transfer ($\frac{\mathcal{E}_i}{I_i+R_i}$) are prioritized. This approach increases the efficiency of using the mobile devices because it increases the probability of keeping device CPUs busy with tasks that require a lot of compute power while buffering new tasks at the controller. (2) To maximize the useful computation and increase processor utilization, the task is assigned to the mobile device that enables getting its results earlier regardless of the time taken to send the results of previously assigned tasks as long as (i) it will be able to send the results before leaving the cluster and (ii) it maintains the feasibility of receiving the results of the previously assigned tasks. (3) To maximize the amount of tasks executed by the cluster, the controller assigns as many tasks as it possibly can before a results gathering event is triggered by our *results gathering heuristics*.

Results Gathering Heuristics: Determining when to start gathering the available results from the devices is a very important question. Premature gathering of results wastes an opportunity of sending more tasks to the executing nodes and increasing computational throughput. Late gathering, however, risks wasting a portion of the results and having to reassign some incomplete tasks, which decreases the computational throughput. Therefore, we adopt two mechanisms while gathering results: (1) essential gathering mechanism and (2) early gathering mechanism.

The essential gathering mechanism clusters the results that have to be transmitted together and sends them in the following case:

$$\mathcal{T}_{\text{remaining}} < (1+\alpha)\mathcal{T}_{\text{needed}}$$

where $\mathcal{T}_{\text{remaining}}$ is the remaining time before the deadline, $\mathcal{T}_{\text{needed}}$ is the needed time to completely send these results to the controller, and α is a safety factor determined by the controller based on how accurate its estimates are about the available bandwidths and the departure times. We highlight that once the essential gathering event is triggered, we use "earliest deadline first" heuristic to gather the clustered results.

The early gathering mechanism utilizes the network in case of the absence of feasible assignment of new tasks to obtain the results. This approach keeps gathering one-task result at a time from the available results with the soonest deadline, until a new task arrival occurs or a change of the system status and parameters takes place.

IV. Evaluation

In this section we evaluate the performance of the Femto-Cloud system, We start by describing our experimental setup followed by presenting and analyzing our results.

A. *Experimental Setup*

Devices	Computation Capacity
Galaxy S5	3.3 MFLOPS
Nexus 7 [2012]	7.1 MFLOPS
Nexus 7 [2013]	8.5 MFLOPS
Nexus 10 [2013]	10.7 MFLOPS

TABLE II
Experimental Device's Characteristics.

Parameter	Values
Chess input size (MBytes)	[0.5, $\underline{2}$, 16]
Average user arrival rate (user/min)	[$\underline{2}$, 8]
Average user presence time (min)	[0.25, $\underline{2}$, 5]
Average device's available bandwidth (Mbps)	$\underline{20}$
Average presence error ratio (%)	[-50, $\underline{0.0}$, 50]

TABLE III
Experimental parameters. The underlined values are the defaults.

To have a realistic performance evaluation, we start by identifying the available capacity in real mobile devices, and the compute requirements of real applications. First, we conduct a measurement study running a matrix multiplication application, we develop, with different preset computational loads (MFLOPs) on a set of mobile devices. We summarize the results of this study in in Table II, which shows the average background thread capacity for the mobile devices. We conduct another measurement study to determine the compute resource usage of different real applications. Table IV summarizes this study and shows the compute resource usage of the following three applications: (1) Chess game in high difficulty mode, (2) a video game called Angry Bird Space, and (3) Object recognition in a video feed (Video Processing). In our evaluation, we use the results of these studies coupled with a newly defined compute intensive application.

Tables II, IV, and III summarize our experimental parameters. Throughout our evaluation, we use a Poisson arrival process to model the arrival of new users as well as the arrival of new tasks. We use the following performance metrics:

- **Computational Throughput:** This is the average amount of useful computations finished by our femtocloud per second (MFLOPS).
- **Compute Resource Utilization:** This is the average utilization of the compute resources in our cluster. To calculate this utilization, we only consider useful computations, which belong to tasks completed by femtocloud.
- **Network utilization:** This is the average busy time of the network for sending tasks or receiving results.

Overall, we conduct two different sets of experiments. The first set of experiments (Section IV-B) aims for understanding the effect of different environmental parameters on the performance of femtocloud. In these experiments, we simulated different environments and studied the effect of different parameters in the performance of femtocloud. The second set of experiments (Section IV-C) sheds light on the performance of our developed prototype.

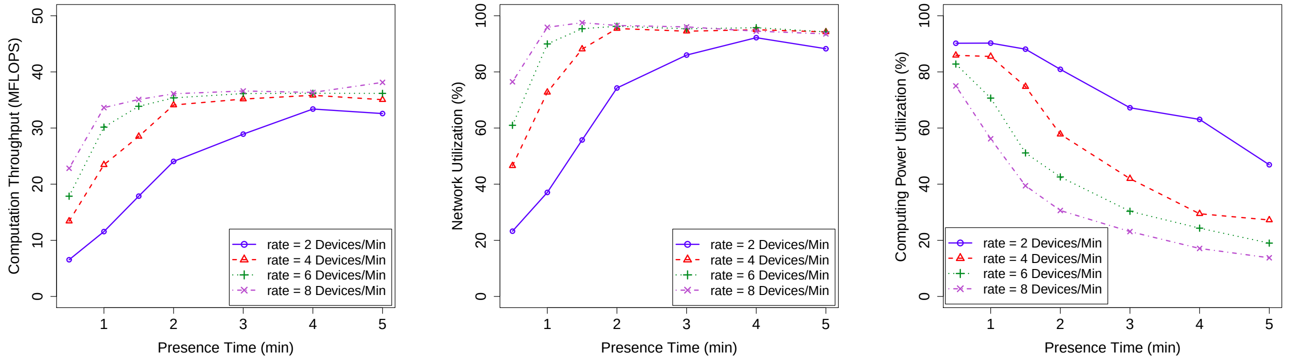
B. *Femtocloud Simulation Results*

In this section, we study the impact of changing different environmental parameters on the performance of femtocloud. We start by studying the impact of user arrival rate and presence time followed by the true effect of stability in the system. We also study the impact of changing task characteristics and robustness to estimation errors. In a subset of these experiments, we compare femtocloud against a presence

Task Type	Input	Computation	Output	arrival rate
Chess	2 MBytes	10 MFLOPs	0.2 MBytes	1 task/sec
Video Game	0.2 MBytes	30 MFLOPs	2 MBytes	2 task/sec
Video Processing	3.125 MBytes	60 MFLOPs	1 MBytes	1 task/sec
Compute Intensive	8 MBytes	100 MFLOPs	0.5 MBytes	0.5 task/sec

TABLE IV

EXPERIMENTAL TASKS CHARACTERISTICS AND EVALUATION PARAMETERS.



(a) Computational Throughput

(b) Network Utilization

(c) Computational Resource Utilization

Figure 3: Impact of changing device arrival rate and presence time.

time oblivious scheduler (PreOb). Such scheduler uses the same task assignment heuristic used by femtocloud but without taking the presence time of a device into account. Due to its unawareness of the presence time, it requests the results from the device one they become available.

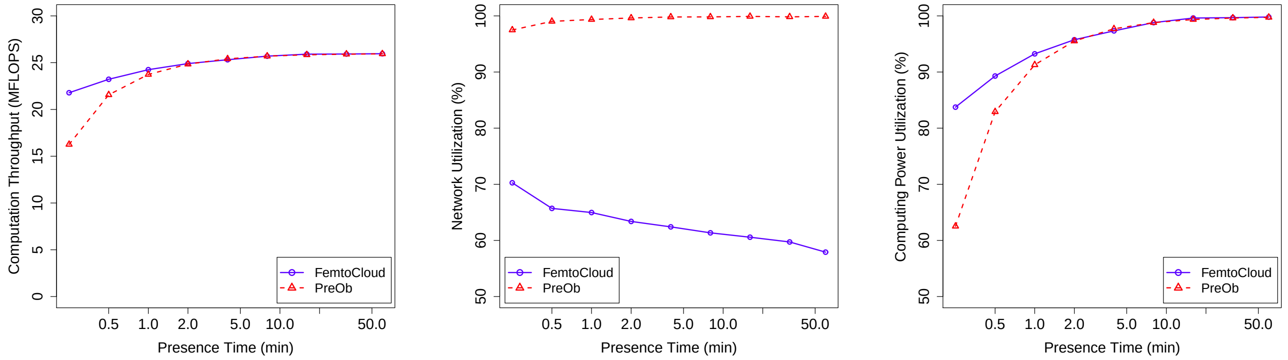
Impact of changing user arrival rate and presence time: Figure 3 shows the effect of changing the average user presence time and the average user arrival rate on the performance of femtocloud. Figure 3 (a) shows that the increase in the presence time or the user arrival rate, significantly enhances the performance and increases the femtocloud's computational throughput. This increased computational throughput saturates for large values of the arrival rate or the presence time. To explain the reason behind this saturation, we refer to Figure 3 (b), which clearly shows that the network utilization increases as more tasks get assigned to our devices until it becomes highly utilized and unable to support more task assignments.

Figure 3 (c) shows that the devices' utilization decreases with the increase of the presence time or the arrival rate. This decrease is due to having a lot of available devices in the system which enables distributing the load on them and minimizing their utilization and overhead.

Stability Impact: Guided by the derivations we draw from Figure 3(c), it is critical to understand the true impact of the increased user presence on the system independent of the changes to the available compute resources. Therefore we construct an experiment in which we fix all the parameters in the system except the presence time of the devices. In this experiment we have three devices (a Nexus 10 and 2 Nexus 7 devices) and we change the average user presence time from 15 sec to 1 hour. To isolate the effect of presence time, once a device leaves our cluster an identical copy arrives and joins the cluster. Figure 4 shows the results of these experiments and compares femtcloud to the presence time Oblivious scheduler (PreOb). Figure 4 (a) and Figure 4 (c) shows that with the increase of the presence time, both algorithms utilizes the stability to gain more performance. femtocloud's awareness of the presence time enabled it to achieve higher performance than PreOb for low presence time values. Figure 4(b) shows that the femtocloud's increased performance comes with lower network utilization. The main reason is that without the knowledge of the presence time, PreOb assigns tasks to devices that may not be able to execute them and, thus, it may have to reassign them again to another device. This behavior keeps the network unnecessary busy. Figure 4 (b) also shows that with the increase of presence time femtocloud becomes able to execute tasks that require high compute resource and low network usage. Therefore, the more stable the devices in the femtocloud the less it consumes from the network resources.

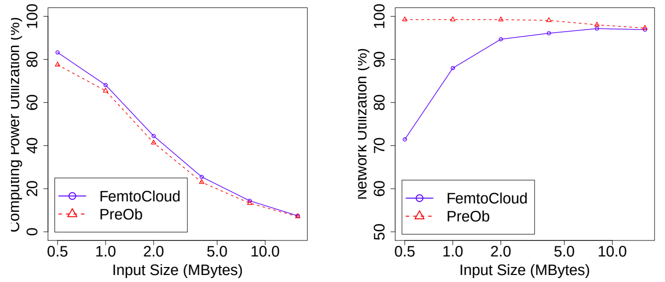
Task characteristics impact: To study the impact of changing the task characteristics, we conduct an experiment that has only single type of tasks (Chess). While maintaining the average computational requirements and average output size as constants, we vary the input size from 0.5 MBytes to 16MBytes. Figure 5 shows the impact of increasing the task input size on the performance of femtocloud. It is clear that with the increase of the input size, the task characteristics moves from being CPU bounded, which enables increasing the compute resource utilization, to be fully Network bounded. Therefore the compute resource utilization decreases and the network utilization increases.

Robustness to estimation errors: To measure the impact of errors in estimating the presence time of the user on the system, we conduct an experiment in which we introduce an Gaussian error and changed the mean from -50% of the presence time to +50% of the presence time. Figure 6 shows that when the error mean is 0, femtocloud is able to achieve the highest utilization of the available devices. When the error is negative, we have a conservative estimate about the presence time which limits femtocloud usage of a device, which leads to decreasing the compute and network utilization. When the



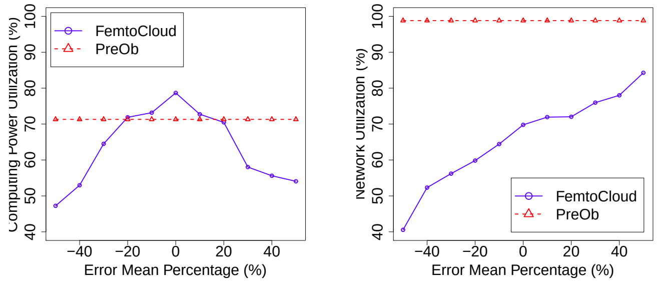
(a) Computational Throughput (b) Network Utilization (c) Computational Resource Utilization

Figure 4: Stability impact.



(a) Computational Utilization (b) Network Utilization

Figure 5: Task characteristics impact.



(a) Computational Utilization (b) Network Utilization

Figure 6: Robustness to estimation errors.

error is positive the computational utilization degrades because femtocloud fails to gather all the executed task results. Note that our early gathering heuristic is responsible for minimizing this effect. Figure 6 (b) shows that the network utilization keeps increasing because femtocloud keeps assigning tasks more and more tasks while moving from a conservative estimate to a less conservative one. Further more, when the error becomes positive, the network utilization will further increase due to reassigning tasks after a device leaves without sending their results.

C. Femtocloud Prototype Evaluation

In this section, we discuss the results we gathered while using our prototype. In our experiment, we use three devices, a Galaxy S5 running Android 4.4.4 in addition to a Nexus 10[2013] and a Nexus 7[2013] tablets running Android 5.0.2. In this experiment, we compare the performance of femtocloud to an oracle which assumes accurate knowledge of all connectivity and execution time for every task on every device. Since this oracle is impossible, we gather measurements from all the devices and use after the fact analysis get the results.

In our experiment, we compare the achieved compute throughput by the oracle and femtocloud under two scenarios: (1) Full presence scenario, and (2) Emulated arrival/departure scenario. In the first scenario, we assume that the three devices existed during the whole period of experiment (1 hour). The main goal of this scenario is comparing the maximum achievable performance of femtocloud to the one achieved by the oracle. In the second scenario, we emulate average presence time of two minutes for each device. We emulated the arrival of new devices by returning the device to the cluster after average of one minute from its last departure. Table V summarizes these experiment results and shows that femtocloud achieved more than 85% of what the oracle achieved in both scenarios.

Scenario	Oracle	femtocloud
Full presence	16.54 MFLOPS	14.23 MFLOPS
Emulated arrival/departure	10.31 MFLOPS	8.86 MFLOPS

TABLE V
PROTOTYPE PERFORMANCE MEASUREMENTS

V. Related Work

We focus our discussion of related work primarily on systems and architectures to realize offloading of computation from mobile devices. We then briefly mention key supporting technologies.

Early research efforts on offloading computation from mobile devices focused on offloading to the cloud, as exemplified by the MAUI [4], CloneCloud [3], and COSMOS[16] systems. MAUI's primary consideration was to reduce energy consumption at the mobile device, and its primary focus was on enabling fine-grained code offload without placing too much burden on application developers by taking advantage of managed code environments. CloneCloud similarly aimed for ease in enabling mobile applications to offload execution without adaptation, using static and dynamic profiling. COSMOS, however, focused achieving significant execution speedup while maintaining efficient resource allocation and management using elastic clouds. These systems demonstrated combined energy savings and execution speedup for applications with appropriate properties.

During a similar time period, Satyanarayanan and colleagues proposed Cloudlets, introducing a middle tier between mobile devices and the traditional cloud [15]. The driving insight behind cloudlets was that the bandwidth and latency to the traditional cloud would be limiting for certain applications, and thus proximity of resources for offloading was critical. Building on this work, those researchers have recently proposed just-in-time provisioning of these cloudlets for dynamic virtual machine creation so that mobile tasks can be more easily offloaded and executed [8]. Cisco's fog computing architecture bears similarities to cloudlets, from the vision to "extend the cloud computing paradigm to the edge of the network" to the introduction of a layer between data centers and end devices [2]. Cisco positions fog computing as especially suitable for the Internet of Things, where end devices will include a wide variety of sensors and embedded systems in need of infrastructure support, in addition to user devices.

As explained earlier, our femtocloud architecture can be viewed as a re-factoring of the cloudlet architecture into a controller and compute resources, that need not be realized in the same box but instead can leverage idle cycles on local devices. Hints in this direction can be found in Nishio et al. where a "local cloud" is constructed of mobile devices, one of which is elected as a resource coordinator [13]. That work, however, focuses less on architecture and system design and more on developing a unifying utility function that allows an optimization framework for resource sharing under the assumption that the nodes in the local cloud are stable, experiencing no churn.

More generally, our work fits into the class of cooperation-based architectures for mobile cloud computing, as identified by Guan et al. in their survey of mobile cloud computing research [7]. Extreme forms of cooperation, without a controller, map to one end of the spectrum that femtoclouds lie upon, as illustrated earlier. These extreme approaches include systems such as Serendipity [17], Mobile Device Clouds [11], [10] and Hyrax [18].

None of these systems would be possible without significant effort and advances in areas that address key questions of what to offload and how to offload, a division of concerns nicely articulated in Gordon et al. in their work on OS support for offloading [6]. By no means an exhaustive list, these technologies include application profiling, code partitioning, energy and computation estimation, thread migration, virtual machine synchronization, safety and security while running on unmanaged machines.

VI. Conclusion

In this paper, we presented the femtocloud system that we developed to leverage mobile devices to provide cloud services at the edge. This system falls in the middle of the stability spectrum of such clustered systems and is considered an essential step towards opportunistic computing[12]. We presented the design and architecture of the system. We identified the task scheduling problem as an important part of the design of such a system and developed an optimization framework that led us to scalable heuristic solution to the problem. Our evaluation demonstrated the potential for femtocloud clustering to provide a meaningful compute resource at the edge. Finally, we plan to extend this work by (1) enabling our task assignment problem to take mobile devices' energy consumption into account, (2) designing a suitable user incentive system to be used in our femtoclouds, (3) deploying our system in various environments, and (4) conducting thorough evaluation of FemtoCloud in each of these environments.

VII. Acknowledgments

We would like to thank the anonymous reviewers for their insightful feedbacks. This work was supported by the US National Science Foundation through grants NETS 1409589 and NETS 1161879.

References

[1] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang. The case for cyber foraging. In *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, EW 10, 2002.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. SIGCOMM MCC, 2012.

[3] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: Elastic execution between mobile device and cloud. EuroSys, 2011.

[4] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: Making smartphones last longer with code offload. ACM MobiSys, 2010.

[5] J. Flinn. Cyber foraging: Bridging mobile and cloud computing. In *Synthesis Lectures on Mobile and Pervasive Computing*. Morgan and Claypool, 2012.

[6] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen. Comet: Code offload by migrating execution transparently. USENIX OSDI, 2012.

[7] L. Guan, X. Ke, M. Song, and J. Song. A survey of research on mobile cloud computing. IEEE/ACIS ICIS, 2011.

[8] K. Ha, P. Pillai, W. Richter, Y. Abe, and M. Satyanarayanan. Just-in-time provisioning for cyber foraging. ACM MobiSys, 2013.

[9] Y. Kwon, S. Lee, H. Yi, D. Kwon, S. Yang, B.-G. Chun, L. Huang, P. Maniatis, M. Naik, and Y. Paek. Mantis: automatic performance prediction for smartphone applications. In *USENIX ATC*, 2013.

[10] A. Mtibaa, A. Fahim, K. A. Harras, and M. H. Ammar. Towards resource sharing in mobile device clouds: Power balancing across mobile devices. In *SIGCOMM MCC*, 2013.

[11] A. Mtibaa, K. Harras, and A. Fahim. Towards computational offloading in mobile device clouds. IEEE CloudCom. IEEE, 2013.

[12] A. Mtibaa, K. A. Harras, K. Habak, M. Ammar, and E. Zegura. Towards mobile opportunistic computing. In *IEEE International Conference on Cloud Computing (IEEE CLOUD)*, 2015.

[13] T. Nishio, R. Shinkuma, T. Takahashi, and N. B. Mandayam. Service-oriented heterogeneous resource sharing for optimizing service latency in mobile cloud. MobileCloud, 2013.

[14] M. Satyanarayanan. A brief history of cloud offload: A personal journey from odyssey through cyber foraging to cloudlets. *SIGMOBILE Mob. Comput. Commun. Rev.*, 18(4):19–23, Jan. 2015.

[15] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 2009.

[16] C. Shi, K. Habak, P. Pandurangan, M. Ammar, E. Zegura, and M. Naik. Cosmos: Computation offloading as a service for mobile devices. In *ACM MobiHoc*, 2014.

[17] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura. Serendipity: Enabling remote computing among intermittently connected mobile devices. In *ACM MobiHoc*, 2012.

[18] C. Teo. Hyrax: Crowdsourcing mobile devices to develop proximity-based mobile clouds, 2012.