ECOS: Leveraging Software-Defined Networks to Support Mobile Application Offloading

Aaron Gember, Christopher Dragga, Aditya Akella
University of Wisconsin, Madison
{agember,dragga,akella}@cs.wisc.edu

ABSTRACT

Offloading has emerged as a promising idea to allow resource-constrained mobile devices to access intensive applications, without performance or energy costs, by leveraging external computing resources. This could be particularly useful in enterprise contexts where running line-of-business applications on mobile devices can enhance enterprise operations. However, we must address three practical roadblocks to make offloading amenable to adoption by enterprises: (*i*) ensuring privacy and trustworthiness of offload, (*ii*) decoupling offloading systems from their reliance on the availability of dedicated resources and (*iii*) accommodating offload at scale. We present the design and implementation of ECOS, an enterprise-centric offloading framework that leverages *Software-Defined Networking* to augment prior offloading proposals and address these limitations. ECOS functions as an application running at an enterprise-wide controller to allocate resources to mobile applications based on privacy and performance requirements, to ensure fairness, and to enforce security constraints. Experiments using a prototype based on Android and OpenFlow establish the effectiveness of our approach.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/Server*

General Terms

Algorithms, Management, Performance, Security

Keywords

Offloading, Mobile devices, Energy savings, Enterprise network

1. INTRODUCTION

Mobile devices such as smartphones and tablets are being increasingly recognized as critical business tools, and enterprises are targeting both specialized and common mobile applications to these platforms [23]. Unfortunately, the complexity and overhead of the applications [15, 17], and the accompanying security issues, are seen as major impediments to full-fledged deployment on mobile devices [4].

Application-independent offloading has long been recognized as an important mechanism for enabling smartphone users to access resource-intensive applications without incurring energy and performance costs [9, 13, 15, 22, 25]. As mobile devices become the primary platforms for some employees, we believe mobile application offloading will be essential for running resource-intensive enterprise applications—e.g., modeling and analysis tools, handwriting and speech recognition, etc.—with suitable performance and energy usage. Moreover, this need will persist for the foreseeable future as device demands continue to outstrip battery capabilities [8]. However, two key roadblocks currently prevent enterprise adoption of mobile application offloading.

1. Privacy and trust: Enterprise applications frequently operate on data with strict privacy requirements, requiring the use of trusted resources (e.g., servers in a local data center) for application execution. The majority of offloading systems ignore such privacy requirements, selecting compute resources solely based on connectivity characteristics and processing capabilities [14]. Even systems which are capable of limiting execution to specific compute resources [9] are insufficient, as they overly restrict offloading opportunities and may unnecessarily impose energy and latency costs.

2. Resource sharing and churn: Enterprises may have thousands of employees using mobile devices, all of which may desire offloading simultaneously. While existing systems address *what* and *how* to offload from a single device [15, 22], no attention has been given to the effects of many devices with different objectives simultaneously offloading to the same compute resources. The energy and latency benefits of offloading assumed by some frameworks to be fixed [14, 15] will, at enterprise scale, be quite dynamic. This dynamism increases even more when considering the range of potential compute resources available—idle desktops, local servers, public clouds—and the changes in capacity and availability that accompany this diversity.

We present an enterprise-centric offloading system (ECOS) that can be coupled with existing offloading frameworks to address the above roadblocks. ECOS is based on two observations: (*i*) There are plenty of idle resources available in enterprise networks; our unique measurements of resource availability in a campus network confirm this (§4). (*ii*) Tight administrative control over compute and network resources in enterprises provides the means for mobile applications to access trusted resources, enabling natural mechanisms to ensure privacy and trust. Thus, the central design guideline in ECOS is to allow many mobile application offloads to opportunistically leverage idle compute resources, while tightly controlling the locations where specific applications are offloaded depending on trust, privacy and performance constraints of different users and applications.

ECOS leverages *Software-Defined Networking (SDN)* to meet this guideline. To the best of our knowledge, this is the first attempt at using SDN to better meet the demands of mobile applications. ECOS functions as an application running at an enterprise-wide controller that orchestrates all mobile application offloads. The controller application: (*i*) enforces trust and privacy constraints—specified using a simple, expressive policy language—by tightly controlling the flow of traffic between mobile devices and selected compute resources and by triggering additional higher-layer security mechanisms as necessary; and (*ii*) uses fine-grained resource management algorithms to exercise control over an enterprise's network, desktop, cloud, and mobile device resources and *guarantee* the desired benefit in terms of latency improvement, energy savings, or both. While our centralized framework is an extreme point in the design space, we claim that simultaneously meeting the privacy, trust, and resource constraints identified above, while optimally supporting mobile offloading at scale in enterprises, necessitates this choice.

Key challenges arise in designing ECOS. First, we show that securing offload adds non-trivial energy and latency during both connection setup and state transfer. Hence, careful choices must be made in deciding whether to offload an application that requires privacy and also in designing offload schemes to control the overhead. Second, because a limited number of compute resources are shared by a variety of mobile applications with differing performance, energy and security requirements, we must design clever allocation algorithms that (*i*) adapt quickly to diverse application demands and changing resource availability, (*ii*) ensure applications see equitable and substantial benefits, and (*iii*) control the impact on regular desktop applications. Third, our approach should minimize the amount of work mobile devices undertake and shift a majority of the decision-making from the devices to the controller. Finally, the controller, where the algorithms run, must offer high offload request throughput and low latency. We describe our solutions in §3 and §4.

We have prototyped ECOS using OpenFlow [21] and Android [5]. We evaluate our prototype using two mobile applications that are representative of enterprise workloads. Using 12 phones and up to 6 desktops, we measure the benefits ECOS can provide in a small enterprise setting where phones have varying goals and privacy constraints. In all cases, application latency improves by as much as 94% and energy savings can be up to 47%. In addition, the amount of execution state applications need to send can be reduced by up to 98% for some applications by employing resource affinity and maintaining execution state on compute resources, further improving benefits.

We summarize our contributions as follows:

- We analyze the overhead of transport-layer encryption and categorize the risks associated with enterprise data. Based on these observations, we design (*i*) a simple, expressive policy language that captures privacy constraints of applications/devices and trust levels of resources, (*ii*) a decision process for applying encryption, and (*iii*) network-level policy enforcement mechanisms.
- Using measurements of resource availability from an enterprise-like setting, and our policy language, we design algorithms for allocating resources and managing offloading state at these resources in a way that provides equitable and desirable benefits.
- We prototype our system using Android and OpenFlow/NOX. We conduct several experiments using our prototype, illustrating that application latency improves by up to 94% and energy savings can be up to 47%.

2. BACKGROUND

In this section, we discuss: (*i*) prior proposals for offloading and how ECOS augments them, (*ii*) when ECOS can most help enterprise applications, and (*iii*) the design requirements to ensure ECOS is practical and useful. We conclude the section with an overview of ECOS.

2.1 Prior Offloading Proposals

In offloading, parts of a mobile application are run on a different compute resource to offer improved performance, lower energy usage, and/or higher utility to a mobile device user. Many offloading systems have been developed in the past decade, with somewhat different goals. AIDE dynamically partitions memory-demanding mobile Java applications, minimizing the required communication between the mobile device and the compute resource [22]. Chroma uses developer-specified execution strategies (i.e., tactics) to divide execution of code modules with varying complexity and accuracy between local and remote resources; a tactic is selected at runtime based on currently available mobile device and server resources. [9, 18]. MAUI offloads methods from .NET applications to a remote runtime environment based on a history of energy consumption [15]. CloneCloud uses function inputs and an offline model of runtime costs to dynamically partition Android applications between a weak device and the cloud, with the goal of increasing performance or improving failure resiliency [14, 13].

None of the proposals directly addresses the privacy requirements of offloaded applications. Chroma provides some notion of resource trust [9], but with limited flexibility. Alternative methods of augmenting a mobile device's capabilities require the use of specialized APIs [26, 32] or complex trust establishment schemes [25]. Moreover, existing proposals focus on *what* and *how* to offload from a single mobile device and do not consider the effects of multiple offloads sharing compute resources. ECOS can be coupled with any of the offload mechanisms above to overcome these limitations; our implementation (§5) extends a hybrid of Chroma and CloneCloud. However, the benefits ECOS offers may be different from prior systems, especially when applied at scale and when privacy is considered. We explain this next.

2.2 Application Benefits

Four properties allow both current and future mobile applications to potentially benefit from ECOS.

(P1) Significant computation. Offloading requires processing time on mobile devices to capture execution state and time to transfer it over the network. This overhead must be small enough to not offset the speedup from offloading. Thus, applications with significant compute blocks are the most likely candidates to observe *latency benefits*.

(P2) Small amounts of state exchange. The amount of execution state transferred from the mobile device should be small so the *energy cost* of wireless state transfer, which is known to be significantly more expensive than CPU usage [10], does not exceed the energy savings from offload.

The precise computation size and state size required for offloading to be beneficial depends on the quality of the network link and the processing speed of the compute resource, both of which are considered in existing offloading models [13, 20]. However, in ECOS, the security sensitivity of applications and the level of multiplexing on compute resources also influence whether offloading is beneficial.

(P3) Security sensitivity. Security-insensitive applications can run on any compute resource (idle desktop, local server, or public cloud) and require no higher-layer security; hence their performance is driven by **P1** and **P2**. In contrast, some applications are limited in the resources they can leverage—e.g., applications where data should not leave the enterprise premises due to legal issues can never run on public clouds—or applications may require additional security mechanisms—e.g., applications computing on private user data should always use encrypted communications. If no available enterprise resources provide high enough trust for such applications, then offloading is not possible. Similarly, if the latency and energy overhead of encryption—a result of additional CPU cycles and increases in state size—is too high, then the applications cannot benefit from ECOS.

(P4) Resourcing multiplexing. Because ECOS opportunistically leverages compute resources, unlike prior systems, it is possible that offloading saves energy but does not improve latency. This can happen, e.g., when *multiple* apps each satisfying **P1** and **P2** are offloaded to the same desktop. Of course, like prior systems, ECOS can also result in both latency and energy benefits, or latency benefits alone (for apps satisfying **P1** but not **P2**, e.g., face recognition).

Speech-to-text is a compelling example enterprise application that satisfies these properties. (P1) Analyzing the audio stream requires significant computation. (P2) The audio data is limited in size. (P3) Dictations may be confidential, requiring the data and its processing to remain within the enterprise. (P4) Reasonable amounts of delay can be tolerated.

2.3 ECOS Design Requirements

In considering the latency, energy, and security constraints of mobile applications in a large-scale enterprise setting, the ECOS framework must, ideally: (*i*) Know for a given application if the user is expecting energy or latency savings from offload; (*ii*) Identify if applications are security sensitive, pick candidate compute resources accordingly, and provide encrypted channels for such applications; (*iii*) Assign resources so the overall energy and latency benefits are significant, and benefits are equitably distributed across mobile devices and usage scenarios (e.g., privacy-sensitive vs not); (*iv*) Adapt dynamically to changing compute resources without impacting offloaded applications; (*v*) Require minimal decision-making involvement from mobile devices.

2.4 ECOS Overview

ECOS orchestrates all offloads using an SDN application running atop an enterprise-wide controller.

Mobile applications desiring offload contact the controller to request resources. ECOS determines what privacy level the mobile application requires and subsequently decides if the choice of compute resources needs to be limited and if data needs to be secured in transit between the mobile device and a compute resource (§3). The controller considers the costs of security relative to expected user benefits to ensure ECOS provides latency and/or energy savings despite the overhead of ensuring privacy. The challenge lies in creating an expressive policy language that allows administrators to exercise tight control over privacy and trust levels for applications, devices and resources.

The SDN application carefully selects a compute resource—among desktops, local servers and public clouds—based on gathered utilization information (§4). If a mobile user wants performance improvements, the controller assigns a resource with plenty of idle CPU time. Knowing exactly which resources to select is complicated by the fact that resource availability changes over time. ECOS prefers to use the same resource for subsequent offloads but can flexibly switch resources as necessary. Another challenge is ensuring fairness in resource allocation and overall efficiency.

While the actual offload takes places, ECOS enforces its security and resource decisions by installing and manipulating network forwarding state (§3.3). Programmable switches give ECOS tight control over communications and allow the mobile device to move within the network during offload. Dealing with rogue offloads is an additional challenge, which ECOS addresses using a network with default-off behavior.

3. SECURITY

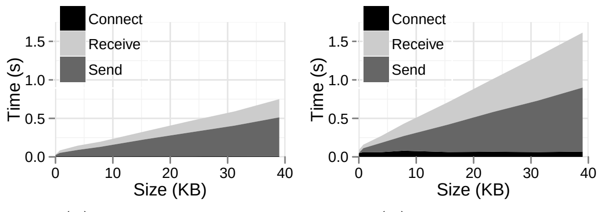
Normally, all execution and associated data stays within a mobile device unless an application explicitly communicates with a third party. Offloading introduces the possibility for data to leave the confines of the mobile device without an application's explicit actions. Thus, privacy and trust (we use the term security to refer to both at once), which are paramount in enterprises, become important concerns.

Two issues arise when accommodating security: (*i*) when to invoke security mechanisms, and (*ii*) if an application needs security, how to determine if it should be offloaded and then, how to secure it. In addressing these issues, ECOS uses two insights. First, not all enterprise applications desire strict privacy; in some cases, it is sufficient if application data is kept within an enterprise boundary. Second, tight administrative control over compute and network resources in enterprises, through SDN, provides a direct way for mobile device users to access trusted resources for offloading, alleviating the need for complex trust establishment schemes.

3.1 Security Risks and Overheads

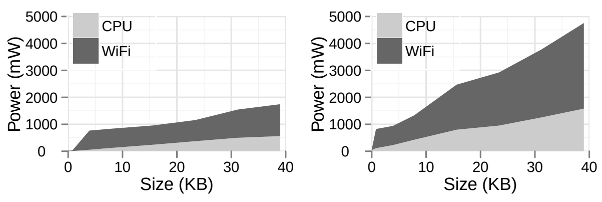
Mobile applications utilize many different types of data. For example, an image recognition application may work with photos taken at an office party, while an optical character recognition application may operate on patient medical records. Some of this data needs to remain confidential, while no risk is posed if other data is viewed by a third party.

Most enterprise data falls into one of three basic categories. *User-private* data should only be accessed by specific users, e.g., a person's medical information can only



(a) Unencrypted (b) Encrypted

Figure 1: Execution state transfer latency



(a) Unencrypted (b) Encrypted

Figure 2: Average power consumed for state transfer

be accessed by individuals working with the patient [6]. *Enterprise-private* data should not be leaked outside the enterprise, e.g., intellectual property such as code and internal memos. *No-private* data can be viewed by anyone, e.g. news releases.

The same privacy restrictions that apply to these categories of data must also be applied to offloads, since an application's execution state likely includes data the application has obtained (or generated). This means security mechanisms must be applied to offloading communications and executions to avoid compromising privacy. Unfortunately, these mechanisms, e.g., transport-layer encryption, can have significant costs.

3.1.1 Latency and Energy Overhead of Encryption

To understand the extent to which basic security mechanisms can influence the benefits of offloading, we measure the time and energy overhead of applying TLS encryption to offloading communications. We compare the overhead both with and without encryption for varying amounts of execution state. Our measurements use an Android emulator [5] and a 2GHz dual-core desktop running an x86 version of Android in a virtual machine [7]. The emulator and VM both run our modified Dalvik runtime environment capable of capturing and loading execution state. (We show in §6.2 that an emulator is reasonable approximation of an actual Android phone.) Each test consists of a single method call which includes several arguments and objects in its execution state but performs no computation.

Latency. The latency overhead of offload with and without encryption is shown in Figure 1. We observe, in both cases, that connection overhead occupies a noticeable portion of the time spent offloading, though it becomes less significant as the size of the offloaded state increases. This can be alleviated by reusing connections. More importantly, the costs of sending and receiving execution state are approximately 40% higher with encryption. Encryption introduces significantly more latency because of the additional control mechanisms, the need to spend processing time encrypting the data, and the higher net volume of encrypted data. For the chess application used in our evaluation (§6), the send (receive) time per-offload increases from ∼0.2s (∼0.1s) without encryption to ∼0.35s (∼0.3s) with encryption.

Energy. Physical energy measurements are difficult due to the limited availability of device schematics and the frequent interactions between individual components. Instead, we take advantage of an existing power model [31] that takes as input fine-grained measurements of CPU utilization, CPU frequency, and the number of packets and bytes sent and received wirelessly. We only include the CPU and Wi-Fi energy components of the model because the other components on a device are typically not impacted by offloading.

Figure 2 shows the results of our power estimates. Energy consumption grows steadily with the size of the offload state for both encrypted and unencrypted connections. CPU energy usage grows roughly linearly, due to increasing costs for deconstructing and reassembling the increasing amounts of state. The overhead from encryption nearly doubles these costs. The increasing Wi-Fi energy usage results from the increasing amount of time required to offload larger state sizes, which grew from a median of 0.1s or less when no state is sent to a median of 1.5s and 0.7s for encrypted and unencrypted connections, respectively, when sending 39 KB of state. During this time, the Wi-Fi interface remains in its high power state, causing it to consume more energy. Conversely, with small amounts of state ($<$ 1KB) transfered over unencrypted connections, the power consumption is very low because the device is able to stay in low power mode throughout the operation.[1] For the chess application used in our evaluation (§6), the energy per-offload used for state transfer increases from ∼950mW without encryption to ∼2470mW with encryption.

Summary. These observations have key implications for the design of ECOS. First, security mechanisms that use mobile device resources, e.g., TLS encryption, should be used only when necessary. Second, the connection setups and data transfers required for offload should be minimized.

3.2 Security Policy

ECOS ensures private data remains secure, while maximizing offloading opportunities and benefits, through the use of a simple, yet expressive, security policy. This policy allows ECOS to minimize the use of resource-intensive TLS encryption and maximize the use of SDN—to exercise tight network control and carefully select compute resources.

The security policy (e.g., Listing 1) conveys (*i*) the privacy level of devices and applications and (*ii*) the trust level of compute resources. Mobile devices and compute resources are identified based on their MAC address, IP address, or primary user. Applications are identified based on a cryptographic hash derived from the executable. ECOS could be adapted to use pre-distributed certificates or user credentials for identifying and authenticating devices and applications.

Mobile devices and applications are labeled as utilizing *enterprise-private*, *user-private*, or *no-private* data. By default, mobile devices are no-private. Devices can be further classified based on who they belong to: e.g. the CEO's device is user-private or all company-owned devices are enterprise-private. Users may also request their devices to be registered as user-private; it is up to administrators to honor such re-

[1]While the additional power consumption in high power mode is significant, this will not be visible if the interface is being actively used by other tasks, causing it to already be in high power mode.

Listing 1 Sample enterprise security policy

```
### Assign privacy levels to mobile devices ###
mobile alice = 00:0F:89:B1:C3:D5 enterprise;
mobile bob = 00:0F:71:6A:17:DF user;
### Assign privacy levels to applications ###
app chess = <8232afd556a9fc56c68cc13113c3f2f5> none;
app speech = <beef35481503415c65555ea068c07ac5> user;
### Assign trust levels to resources ###
resource carol = 192.168.1.10 enterprise;
resource dave = 192.168.1.20 enterprise;
resource cloud = 10.0.0.50 none;
```

quests. Applications are assigned privacy levels based on the most private data they are expected to access. Only if an application has *zero* likelihood of accessing private data is it classified as no-private, e.g., a map application. Applications that are likely to access personal information are labeled as user-private, and all other applications are labeled as enterprise-private (the default label for applications).

We acknowledge that the granularity of these labels is quite coarse. Ideally, an application should be labeled based on the specific data objects it is accessing at a given point in time. To do this, it may be necessary to track the flow of private information on the mobile device [16, 24]. Data with known privacy requirements—e.g., data coming from specific servers, email senders, or file locations—can be tracked, enabling dynamic knowledge of the data contained in an offload. However, approaches to track information flow impose performance penalties (TaintDroid imposes 14% CPU overhead [16]) and are still evolving. Our coarser-grained approach, in contrast, takes a conservative view of privacy and trust. Privacy levels are assigned based on the strictest level of privacy an application may ever require, irrespective of its current data usage. The advantage is that it is far simpler and efficient to implement, and it subsumes privacy constraints based on individual data items and bytes. As will become clear after the description of the security mechanisms (§3.3), the downside of using coarse labels is that it may unnecessarily prevent offloading of some applications.

In ECOS, compute resources are assigned trust levels that mirror the privacy levels: *enterprise-trust*, *user-trust*, and *no-trust*. Resources internal to the enterprise—personal desktops and laptops and local data centers—are labeled *enterprise-trust*. External resources—public clouds—are labeled *no-trust*. *User-trust* resources are special because they are trusted by a specific user or subset of users. For example, if Alice and Bob are both friends, they may consider each others' desktops to provide *user-trust*. Our paper does not address how such trust relationships among users are derived and used to specify user-trust resources for a specific user, but there are several ways for an admin to do so: e.g., configuration by hand, based on observed communication patterns among users [27, 29], or based on existing user groups [28].

3.3 Security Mechanisms

One way to ensure data remains secure is to always encrypt offloading communications and always limit the compute resources used for offload, but this adds unnecessary overhead and artificially restricts offloading opportunities. In contrast, ECOS utilizes the nature of the mobile device and application in determining whether encryption and/or limiting the choice of compute resources are really necessary. ECOS bases its decision on the *strictest level* of privacy required by a device and application pair. Moreover, ECOS examines the costs to determine if offloading should even be performed. In all cases, ECOS enforces its decisions using tight network control. We describe each mechanism below.

Require Encryption. All data and state belonging to the offload must be encrypted between the mobile device and the compute resource when the device or application is *user-private*. ECOS achieves this using TLS encryption at the transport layer. Since the added time and energy overhead (§3.1.1) can significantly decrease the number of situations where offloading is beneficial, we seek to reduce transfer sizes and reuse connections through resource affinity (§4.2).

Limit Choice of Compute Resources. When the application or mobile device is *enterprise-private*, offload can only happen to *enterprise-* or *user-trust* compute resources, and in the *user-private* case only to *user-trust* resources. This severely limits the number of resources available for these offloads, placing them at a potential disadvantage. Our resource management algorithm (§4.2) must therefore ensure that these offloads receive a fair amount of resources.

Control Network Flows. ECOS enforces its encryption and resource selection decisions using SDN. Enforcement is crucial because we must ensure that: (*i*) the offload system cannot be abused to inflict attacks on network links and compute resources or to attack other offloads, and (*ii*) the system cannot be used to compromise privacy and trust constraints. We realize these guarantees by relying on the ECOS SDN application to implement a default-off network, tightly control the path taken by individual offload flows from mobile devices to compute resources[2], and constantly monitor network/compute resources and traffic and filter abusive flows.

In a default-off network, network elements (e.g., switches, wireless APs, hosts, etc.) have no forwarding state by default. Forwarding state is installed by an SDN application on the basis of traffic engineering directives [19, 30] and/or reachability policies [11, 21]. Moreover, forwarding is controlled at fine-granularity, in terms of protocols, ports, VLANs, and source and destination IP and MAC addresses.

ECOS's use of SDN to implement a default-off network and control forwarding of traffic at fine-granularity offers key advantages. First, we can ensure that data is encrypted when privacy is desired and compromises of a device cannot lead to leakage of private information. To do this, the SDN application establishes forwarding state only for the appropriate flows: e.g., when a device or application is user-private, only packets destined for port 8443, corresponding to offloading with TLS in our implementation, are allowed to enter the network from the mobile device; all other traffic from the device is dropped at the first hop router. For enterprise-private devices or applications, the SDN application ensures that traffic does not leave the confines of the enterprise. The SDN application can also perform other checks, e.g., periodically routing offload traffic through a middlebox to check for exfiltration of confidential data.

Second, applications can be prevented from making unauthorized access to compute resources during offload. When unauthorized access is detected, e.g., based on unexpected resource consumption activity on a desktop, the controller can simply delete the corresponding forwarding entries in the network, terminating the offload. Rogue applications that aim to steal resources from legitimate offloaded applications

[2]The rest of the network could use other forms of routing.

can also be easily thwarted: ECOS limits the potential impact of rogue applications on performance seeking offloads by providing the latter a minimum share of CPU cycles. For offloads seeking energy savings, care must be taken to ensure rogue applications do not over consume network resources and inflate the communication overheads incurred by legitimate offloads. The controller constantly monitors for usage spikes on compute resources and network links to detect rogue applications; the controller installs drop rules in network elements to "terminate" these applications.

Finally, SDN provides mobility advantages [11]: As mobile devices move through the enterprise wireless network, paths are updated to keep devices and resources connected.

4. SELECTING RESOURCES

Existing offloading frameworks address *what* and *how* to offload from a single device [13, 15, 22], but they do not consider how multiplexing offloads from several devices on a single compute resource will impact performance and energy benefits, nor do they consider the effects of changes in resource capacity and availability over time. In contrast, ECOS is designed to *opportunistically leverage* multiple available enterprise resources to provide benefits to a network of mobile devices while honoring privacy constraints. In this section, we show that plenty of idle desktops exist in enterprise settings to serve application offloads, and we examine the volatility of these resources to understand the implications for resource scheduling in ECOS. We then show how to leverage these resources. The challenge lies in *optimally and fairly multiplexing* offloaded applications amongst these resources to meet the energy, latency and privacy needs of each mobile device. A further complication arises from the fact that we must assign resources without knowing a priori when other offloads will need specific types of resources.

4.1 Resource Availability

Enterprise networks contain a large supply of compute resources that can be leveraged for offloading. These resources tend to be under tight administrative control, making them relatively secure. Some of these resources can be dedicated specifically for offloading (e.g. servers), while others should only be utilized for offloading when they are not serving their primary purpose (e.g. personal desktops and laptops).

To understand idle desktop availability in enterprise settings, we tracked the resource usage of 325 machines (used by ∼600 users) in the Computer Science department network at a large university; this roughly corresponds to a deployment in a modest-sized enterprise. We queried the CPU load of both user desktops and dedicated servers every 5 minutes over a period of 10 days in April 2010.

At any given point in time, over half of the machines have the majority of their CPU idle. Figure 3 shows the median, average, and 90th percentile user load (normalized to one CPU core) per machine throughout the measurement period. The average user load (i.e. fraction of CPU time used by user processes) usually ranges from 15% to 30%, with the median usually ranging from 0% to 5%. Thus, there are suitable volumes of resources already existing in enterprises to serve the offloading needs of mobile devices.

We also measure how user load changes over time. For each 5 minute interval, we calculate the absolute change in user load (normalized to one CPU core) per machine. Figure 4 shows a cumulative distribution of the absolute changes for all five minute intervals and all machines. We observe that about 42% of the time there is no change in user load between 5 minute intervals, and, over half of the time, the change is less than 0.01. Therefore, we conclude that the change in resource availability is usually small.

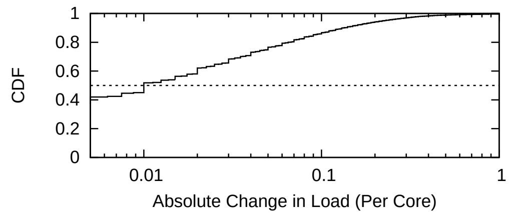


Figure 4: Change in used fraction of processing capacity (per core) over all 5 minute intervals and all machines

Assuming desktop CPUs are 4-5X faster than mobile device CPUs [2], the above results show that a medium-sized enterprise may have enough idle resources to support offload requests for a few hundred mobile applications. Moreover, resource capacity does not need to be tracked at small timescales to make optimal scheduling decisions as the resources are fairly stable; the mechanism for scheduling offloads must mainly worry about churn in the mobile device population.

4.2 Allocating Resources

We now describe how ECOS schedules compute resources to meet application and mobile device security, performance and energy needs in a fair and efficient manner.

In an environment where there are always more suitable compute resources (trusted desktops, servers, etc.) than mobile applications desiring offloading, we can use a simple approach: When a mobile application wants to offload it is assigned a dedicated resource which is not used by other offloaded applications; after an application's offloaded execution completes, a different application can be assigned to the resource. We call this approach *one-to-one scheduling.* Unfortunately, this simple approach is likely to be of limited use. First, we expect that in the future there will be orders of magnitude more mobile applications desiring offloading. The above approach causes some applications to be denied resources and forces them to run on the mobile device. Second, the approach cannot accommodate more complex offloading policies, e.g., parallel offload of application threads [12].

4.2.1 Multiplexing Applications in ECOS

A better alternative when there are limited compute resources is to assign multiple mobile applications to the same resource, an approach we call *multiplexed scheduling.* ECOS relies on a simple heuristic for pairing offload requests with resources in a manner that likely offers the desired benefits.

Input. The input to the heuristic for a given code block b includes the energy savings without encryption n_b, the energy savings with encryption e_b, the estimated execution time on the mobile device t_b, and the mobile device's CPU speed s. §5.2 explains how ECOS obtains these values.

Candidate resources. If an application is seeking energy benefits and the application is deemed to require encryption, then the controller verifies if the energy savings with encryption is positive; otherwise, no resource is assigned and offload is prevented. For applications that either (i) do not require encryption, (ii) have energy savings with

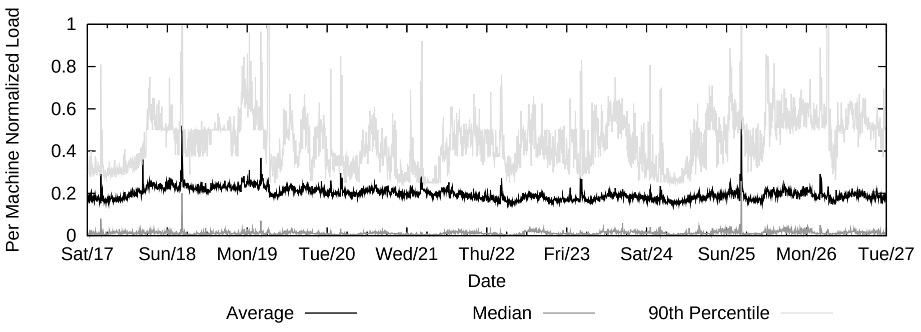


Figure 3: Fraction of CPU used per machine

encryption, or (*iii*) do not care about energy use, a decision must be made whether a resource with sufficient capacity is available.

The time taken, w_b, to execute a code block on a desktop, with or without encryption, depends on the code and the current load on the desktop. For applications seeking performance improvements, we must assign a resource r such that $w_b < t_b$. We can verify this will hold by ensuring the unassigned CPU cycles on r are at least the CPU speed of the mobile device. The available CPU time $a_r = s_r - \sum_m s_m$, where s_r is r's total idle CPU capacity and $\sum_m s_m$ is the sum of the mobile CPU speeds for the applications currently assigned to r. We know $w_b < t_b$ will hold if $s \leq s_r - \sum_m s_m$.

If an application does not seek performance improvements, w_b can be arbitrarily large, meaning that, b can be run anywhere and the application would still see a net energy benefit from b's execution. But care must be taken to ensure that running b does not impact other offloaded applications running on the same resource which care about latency.

Allocation (at the start of offload). If an application is not seeking performance benefits, it is assigned to a resource with other applications only seeking energy benefits. For applications seeking performance benefits, we assign the application to the resource which has the largest a_r at the time the application makes its first offload request. If no $a_r \leq s$, no resources are available and the application is unable to offload; however, it could ask for resources again the next time it desires to offload a particular codeblock. We could assign both performance and energy seeking applications to the same resource, but we separate the two types to avoid the need for complex scheduling mechanisms on the compute resources. If an application seeks both latency and energy savings, we follow the allocation policy for a performance seeking offload, as this places stricter bounds on execution time.

As stated in §3.3, the offload destination depends on the trust level of the application/device: e.g., user-private applications are only offloaded to user-trust machines, whereas no-private applications can be offloaded anywhere. By assigning performance seeking offloads to the machine with the largest available CPU time a_r, we spread the offloads amongst all available resources and increase the likelihood that a sufficiently trusted candidate resource will have available CPU time to serve the privacy constrained offload. In the case of energy seeking offloads, we can assign as many offloads as necessary to a resource, so there is a low likelihood that a sufficiently trusted resource will be unavailable.

Resource affinity. A key issue in allocation pertains to whether the resource used for offload should be changed during the course of an application's execution. Changing means that consecutive codeblocks can be offloaded to different resources. The cost of this is that it requires application state to be re-instantiated at the new resource and new transport connections to be set up. The benefit is that it allows fine-grained scheduling of resources across portions of mobile applications to dynamically adapt to demand and resource availability over time. However, our results in §3.1.1 show that the costs to the mobile device for sending state are significant, and are likely to outweigh the benefits of changing resources. Thus, ECOS avoids this overhead and instead tries to ensure *resource affinity* as much as possible: that is, ECOS seeks to always offload an application to the same resource every time it requests resources. This allows us to keep a partially loaded runtime on the resource to quickly serve future offloads from the same application.

Churn in available resources. Resource affinity relies upon the assigned resource having a constant amount of capacity available during the lifetime of the application, which may not always be true. Conceivably, the user of a desktop could start some resource-intensive task that requires the resources that have been provided to the phone. Our heuristic should not impede the user of a desktop, so we cannot allow the mobile device to continue to offload in such a case. We have several options for how to proceed:

- Deny future offloads from the device until the desktop resources free up. This is simple, but it may not adequately leverage idle resources that may exist in the network and may prevent the device from offloading for a long period.
- Assign the device a new desktop and require it to resend its state. As mentioned before, this could be quite costly to the device, but it may improve the desktop user's experience by allowing state space to be immediately reclaimed.
- Assign the device to a new desktop and have the original desktop migrate the state over the Ethernet the next time the device offloads. This is ideal from the perspective of the mobile device, since it only incurs the cost of establishing a new connection. However, state migration significantly complicates the protocol and may impact the user of the desktop.
- Store state in a network-wide cache after an offload completes. This avoids the need for resource affinity, since any desktop can retrieve the previous state from the cache without the involvement of the mobile device or the original desktop. However, this requires dedicated cache resources.

The best behavior depends on the characteristics of the network. If machines frequently become busy for long periods of time, it may be best to have the mobile device reestablish the state or have the desktops handle state migration. In contrast, if desktops are known to only become busy for brief periods, it may be best to keep the device associated with the desktop and deny its requests until the machine has resources. The controller can track prior usage patterns of a desktop to determine the right behavior.

5. ECOS IMPLEMENTATION

Our prototype implementation of ECOS consists of the following components: (*i*) Compute resource monitors provide resource availability information for offloading resources; compute resources also run a virtual machine (VM) in which offloaded applications are executed; (*ii*) A modified runtime environment (RE) on the phone passes metadata about candidate applications to the controller and performs the actual offload as directed by the controller; (*iii*) An SDN application that runs atop an enterprise-wide SDN controller, orchestrating all offloads and carefully managing the network paths connecting mobile devices and compute resources.

5.1 Compute Resources

Desktops, servers, and remote clouds capable of executing offloaded applications announce their availability and provide an environment for executing the applications. Compute resources provide updates on available CPU cycles. Resource monitors establish a connection to the controller at startup and, based on measurements in §4.1, send XML messages every 5 minutes with the available CPU capacity a_r measured using *mpstat*.

All compute resources also run a smartphone VM to provide an environment for executing offloaded applications. Specifically, we run a native version of Google Android [5]—the phone platform used in our system prototype—in a VirtualBox [7] VM. A *restore agent* in the Android environment manages the offloaded applications running in the VM. Each incoming socket connection from a phone is accepted and passed to a new RE instance which receives the name of the application and all associated execution state from the phone. We assume the application executable already exists on the compute resource; alternatively, applications could be downloaded from the Internet on first offload. The restore agent informs the controller when execution completes.

ECOS gives user applications running on the desktop priority over offloaded applications by over-provisioning for desktop applications: if the collective CPU utilization of desktop applications is u, then ECOS ensures CPU usage of offloaded applications doesn't exceed $a_r = 1 - (u + \delta)$, leaving δ amount of additional compute resources for desktop applications should they need them. We conservatively set $\delta = max(0.1, 0.5u)$.

5.2 Phone Runtime Environment (RE)

The RE on the phone is responsible for (*i*) selecting which parts of an application should be offloaded and (*ii*) transferring an application's execution state to a compute resource.

The RE considers estimated CPU usage, security overhead, and the availability of resources when considering what to offload. We build on ideas from CloneCloud [13] and Chroma [9]. Applications that want to take advantage of ECOS provide a list of computationally intensive methods with the estimated execution time and runtime state size for each method, similar to the specification of tactics in Chroma. The RE uses the measurements from §3.1.1 and the provided state size estimate to approximate the time/energy overhead of offloading with and without encryption. Similarly, the phone approximates the time/energy required to execute the method on the phone, based on the provided execution time estimate t_b and an energy model [31].

If the overhead without encryption is less than executing on the phone, then the phone's *offload agent* asks the controller for resources. The agent connects to the controller when the phone joins the wireless network. An XML message sent to the controller specifies the device (MAC address), a hash of the application binary, energy savings without encryption n_b, energy savings with encryption e_b, estimated execution time t_b, and phone CPU speed s. It also specifies whether the phone desires performance benefits, energy savings, or both, based on user preference, the current battery level, or other factors.

When a method is selected for offload, the method signature, argument values, and any referenced objects must be transferred to the compute resource. Knowing exactly what state to send and where this state resides is challenging. We take a conservative approach to transferring objects, like in MAUI [15]: we transfer objects that are arguments and objects that are referenced by the arguments. After method execution on the compute resource completes, the phone RE must reconcile any changes between the old RE state and the RE state received from the compute resource. A reconciling process compares objects and updates them appropriately, allocating new objects as necessary. Based on the overhead measurements in §3.1.1 and the state similarity measurements in §6.5, ECOS preserves the RE on the compute resource, sending only a delta of the state across different offload points. Computing the delta incurs no more CPU overhead than sending the full state and requires fewer packets. While sending deltas requires additional memory on the phone to store hashes, and preserving state on the compute resource potentially limits the ability of a phone to opportunistically take advantage of available resources, we find that this is a reasonable strategy to use in practice (§6).

5.3 SDN Application

Based on the algorithms described earlier, the ECOS SDN application running on an SDN controller sends an XML message with the IP address of the assigned resource. The phone RE opens a socket (or secure) connection to the resource on port 8400 (or 8443). We implement our SDN application on top of NOX atop a bed of OpenFlow switches [21]. The forwarding state in network elements expires and is flushed out after 1s. Thus, phones with active offloaded applications have to periodically ask the controller for a route (every 0.5s in ECOS) to keep the path to the offload destination alive.

6. EVALUATION

In this section, we evaluate the advantages of using ECOS with a range of experiments to ultimately establish the viability of using ECOS to support enterprise applications on mobile devices. We study the following issues: (*i*) Can ECOS support enterprise applications with different latency, energy and security needs? What benefits do different application classes observe, and what are the costs? (§6.3) (*ii*) To what extent do resource affinity and the ability of ECOS to multiplex several applications on each secondary resource help? Are they ever detrimental in enterprise settings? (§6.4) (*iii*) To what extent do various optimizations to control ECOS's offload overhead, such as preserving state across different code block executions and transferring state under resource churn, help? (§6.5) We describe our setup in §6.1 and §6.2.

6.1 Methodology

We were unable to obtain real enterprise applications due to licensing issues and the lack of source code. Instead, we use two representative applications which can benefit from ECOS: chess, which we use as an (admittedly artificial) stand-in for an compute-intensive AI-based enterprise application that does not seek user privacy (e.g., non-linear decision-making and customer relationship management apps [1]), and speech recognition, which is a stand-in for a speech-to-text enterprise transcriber. The chess game features an AI engine, configured to use three decision iterations. We modified the game to play against itself for 50 moves, with a 10 second delay every other move to simulate the delay of a human. Speech recognition is a computationally and memory intensive application. Unfortunately, Android lacks some crucial audio libraries to run speech recognition, such as CMU Sphinx [3], directly on the phone. Therefore, we model the state size, CPU usage, and memory usage of speech recognition with a mock application to still show the benefits ECOS could provide if phones added the necessary audio support in the future. The application is configured to perform 20 recognitions, with a 10 second delay between tasks. We consider speech recognition to be user-private.

We measure ECOS for a small enterprise setting using a set of Android emulators and desktops. Phone emulators are used in our experiments because we only had access to a single Android developer phone. We show experiments in §6.2 to confirm that the emulators we use are a reasonable substitute for actual phones. Each phone emulator's CPU frequency is scaled to match the behavior of an Android phone in typical under-clocked use. The desktops we use for computational resources are 2.4GHz Intel quad core machines with 4GB of RAM, representative of a typical modern workstation. Our controller runs on a separate machine whose specs are the same as the desktops. The emulated phones and desktops communicate over wired Ethernet. In a real setting, phones would communicate using wireless links of lower speeds; thus, our latency measurements are likely to be an underestimate, but we do not expect the difference to be significant as the amount of state transferred during offload in the above applications is quite small. To estimate power consumption, we measure the number of packets and bytes sent, CPU usage and wireless NIC usage and plug them into energy models [31], similar to our study in §3.1.1.

6.2 Emulator/Phone Comparison

We first perform a set of small-scale experiments to affirm that Android emulators are a reasonable substitute for an actual Android phone. Our setup consists of a single desktop and a single emulator or phone. We run both example applications with and without offloading, measuring the total energy usage and total runtime.

The execution time of the chess and speech recognition applications without offloading are shown in Figure 5 for both the Android emulator and an Android developer phone. The applications run about 20% faster on the emulator, but the

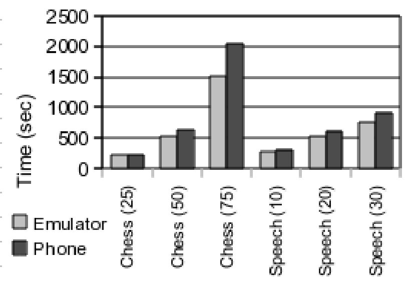


Figure 5: Comparison of application execution times without offloading using a phone versus an emulator

Phone	Application	Goal	Trusted Resources
P1,P2,P3	Chess	Latency	All
P4,P5,P6	Chess	Energy	All
P7	Speech Recognition	Latency	D1, D2, D6
P8	Speech Recognition	Latency	D2, D3, D6
P9	Speech Recognition	Latency	D1, D3, D6
P10	Speech Recognition	Energy	D1, D2, D6
P11	Speech Recognition	Energy	D2, D3, (D6)
P12	Speech Recognition	Energy	D1, D3, (D6)

Table 1: Configuration for applications used

scaling is consistent for both applications with varying numbers of moves/recognitions. We polled the CPU and memory utilization during application execution in all cases at frequent intervals. The CPU ran at 100% utilization whenever the application was busy executing in the case of both the phone and the emulator. Memory usage was also identical. We found the CPU frequency to be similar, although the emulator was running at a slightly higher speed.

We then examined how offload would function. We compared the packet stream to and from the remote desktop when using the phone versus the emulator, and found them to be identical in terms of the number of bytes and packets. Also, in the case of the phone, we did not observe a significant change in the bitrate during the course of any offload. The only difference was a longer offload state transmission time on the phone due to the slower link speed and higher loss rates of a wireless link: for 50 chess moves the total time spent transmitting state over wireless is 10s versus 2.9s over wired, and for 20 speech recognitions the total transmit time is 12s versus 2.3s. We omit the detailed results for brevity.

In effect, these measurements show that the emulator provides a reasonable approximation of execution and offload on a physical phone. The data we collect on the emulator to help estimate phone power drain also provides a qualitatively similar estimate to real power drain on the phone (modulo the accuracy of the model in [31]).

6.3 Full System Analysis

We present an analysis of ECOS for a small enterprise setting consisting of 12 phones (P1-P12) and 4-6 desktops (D1-D6). A mix of both chess and speech recognition run on the phones, with varying goals and privacy levels. Table 1 shows the application specification for each phone. The constraint of only having 4 desktops to serve 12 phones stresses ECOS, but we find it still benefits most applications. With lesser contention when 6 desktops are available, the benefits from ECOS become more significant and equitable.

In these experiments, we assume the desktops are not running any other applications during the entire test du-

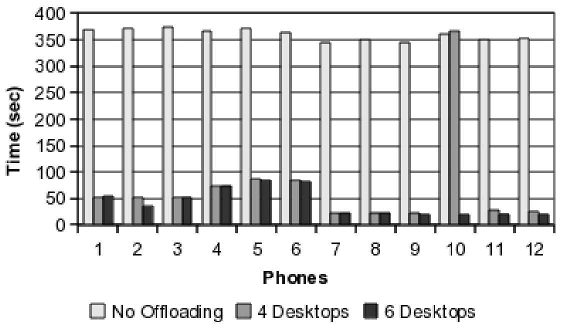


Figure 6: Comparison of application execution times

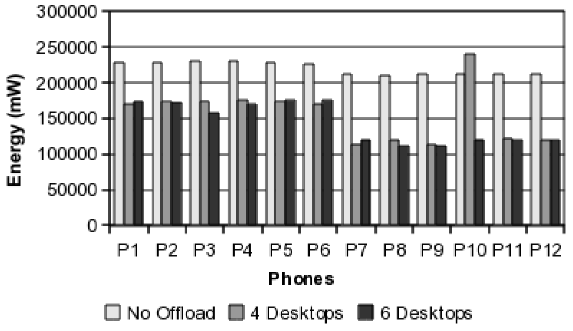


Figure 7: Comparison of application energy usage

ration. Furthermore, the controller multiplexes assignments and enforces resource affinity. Later in this section, we consider other approaches to assigning resources—one-to-one scheduling and no-resource-affinity—in §6.4.

Latency. The execution time (excluding delay between moves or recognitions) for each phone is shown in Figure 6. Without offloading, all applications take approximately 350s to execute. Multiplexing the applications amongst four computational resources significantly reduces the execution time to between 22s and 87s for all phones except P10. The speech recognition application on P10 receives no performance benefit because there is no desktop it trusts that is available to serve applications seeking energy savings. Furthermore, phones P4-6 see nearly 50-60% higher latency than P1-3 although they are running the same chess application. This is because P4-6 requested energy savings, while P1-3 requested latency improvement.

Using 6 desktops provides enough resources leading to all phones seeing better latency than without offloading and 10-20% better net latency compared to using 4 desktops. Furthermore, P10 is able to offload to a user-trusted resource, and, with lower resource contention per desktop, P10's speech recognition application is able to run in as little as 20s.

Energy. Our experiments show that ECOS also offers energy benefits. Figure 7 shows the total energy used by the applications for the same scenario. The energy savings for both performance seeking and energy seeking applications ranges from 24% to 44% with 4 desktops and 23% to 47% with 6 desktops. Again, with 4 desktops, resources are constrained and ECOS is unable to provide energy benefits to P10. Increasing the number of available compute resources allows P10 to attain energy savings equivalent to its peers.

Although not present in these scenarios, in some cases ECOS imposes energy cost for applications that have explic-

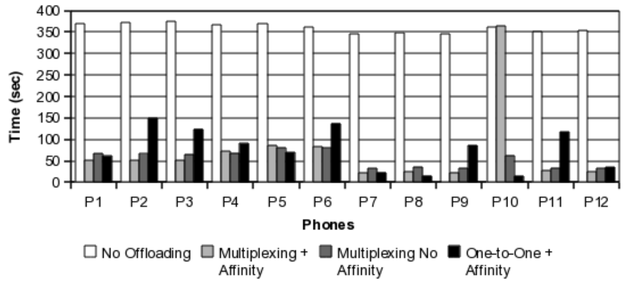


Figure 8: Total execution time using alternative resource allocation approaches

itly requested low latency. Likewise, some applications desiring energy savings may see a degradation in latency compared to no-offload when there is high resource contention. Since our system is opportunistic, the presence of this cost is highly dependent on the application workload.

6.4 Resource Allocation Efficiency

Multiplexing applications on compute resources is one approach to assigning compute resources. However, §4.2 also discussed one-to-one scheduling—assigning a single application to a compute resource at a time—as an alternative method for assigning resources. We compare these two scheduling approaches, both with and without resource affinity. We use the same experimental setup of phones and desktops that was described in §6.3. We measure the total time and energy saved by applications for one-to-one-scheduling with affinity, multiplexed scheduling with affinity, and multiplex scheduling without affinity.

Figure 8 shows the execution time for each phone with the three approaches. First, we observe that one-to-one scheduling results in less offloading opportunities and higher execution latency for two-thirds of the phones. In some cases, e.g. P2, execution takes more than twice as long. This behavior results from the inability to serve more than 4 applications (as many desktops as we have) at any given time. At the same time, applications typically take less time to execute a given offload instance since compute resources are not shared with other offloaded applications. P10, for example, executes the fastest with one-to-one scheduling because it gets full use of the CPU when it is assigned to a desktop.

Second, we observe that for most phones, the resource allocation approaches that use affinity result in lower total latency. This decrease in latency stems directly from the decrease in execution state that must be transferred as a result of preserving execution state at the same desktop. However, avoiding affinity can help provide a fair sharer of benefits when the number of compute resources are limited: e.g., P10 receives significant benefit when using multiplexing with no-affinity as there is more churn in assignments and a greater opportunity for being allocated resources.

6.5 Preserving State Across Offloads

As discussed in section §5.1, ECOS preserves state on compute resources between offloads, decreasing the amount of state that must be transferred during subsequent offloads. This can be especially beneficial when security requirements force data to be encrypted.

We analyze the feasibility of preserving state by measuring how much the state changes between subsequent offloads of the chess application. Figure 9 shows the fraction of state

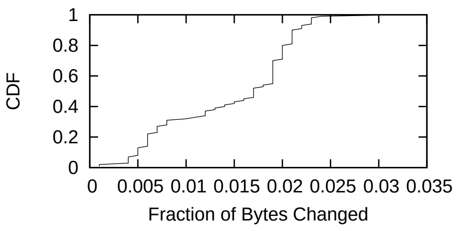
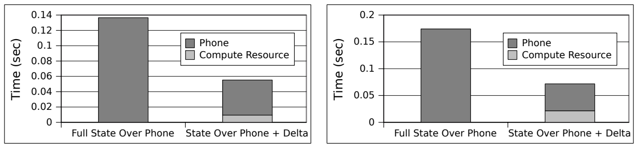


Figure 9: CDF of fraction of state changed between subsequent offloads of chess AI



(a) Unencrypted connection (b) Encrypted connection

Figure 10: Time to transmit state

that changes between subsequent offloads for the first 100 offloaded moves. At most 3.5% of the approximately 37 KB of state changes between offloads. The majority of offloads have only a 0.5% to 2% difference between the current and previous state. The amount of state variation is highly dependent on the specific application, but these results show that preserving state for some applications can significantly reduce the state that must be sent.

Preserving state between offloads can significantly reduce offloading overhead, but it also requires assigning applications to the same compute resource every time. If a compute resource no longer has idle capacity for offloading, the state must either be (*i*) sent in full from the phone or (*ii*) transfered from one compute resource to another. We already measured the overhead of sending the state from the phone in §3.1.1. Figure 10 compares the cost of resending all state from the phone versus transferring state between compute resources and the phone only sending a state delta for subsequent offloads for unencrypted and encrypted connections. For both types of connections, the latter method is more than twice as fast, with the bulk of the overhead occurring on the phone due to connection set up and state transfer. Thus, it is best to handle state transfer between compute resources directly.

In summary, we find that ECOS can support multiple applications with different performance and security needs, offering significant, equitable benefit at low cost. Our design choices have proven crucial for good overall performance, especially when opportunistically leveraging resources, and under encryption: Resource affinity appears to be essential to avoid the overhead of state transfer and connection cost, and ensure good offload performance especially for encryption. For similar reasons, preserving state across offloads and transferring state across connected desktops (when moving offloaded applications) is crucial. Multiplexing several smartphone applications on each desktop is important to ensure multiple applications observe equitable benefits.

Our controller offers high throughput for computing resource allocations, similar to what was observed in [11]. The latency between the time an offload request is made until the time the offload actually begins (which depends on computation of the resource assignment, computing a network path and establishing forwarding state along the path) is negligible. We omitted these benchmarks for brevity.

7. CONCLUSION

We presented ECOS, an enterprise-based offloading system designed to address the security needs of mobile applications and opportunistically leverage available compute resources. ECOS extends the offloading decision process to take into account security requirements and costs, and in doing so, ECOS leverages the unique advantages that Software-Defined Networking (SDN) provides. In ECOS an enterprise-wide controller assigns (trusted) compute resources to applications based on resource availability, administrator specified security policies, and the performance or energy savings goals of mobile devices; ECOS also strictly enforces the security constraints through careful control of the network. We showed that ECOS provides both latency and energy benefits to both no-private and user-private applications.

The main contributions of our work are to show: (*i*) how to accommodate trust and privacy considerations in offloading without resorting to complex and error prone trust schemes and (*ii*) how to scale offloading to many mobile devices and compute resources, and opportunistically leverage these resources. By addressing these issues using SDN, we believe that we have paved the way for wider-spread adoption of offloading to assist current and future mobile applications in the enterprise context.

8. ACKNOWLEDGEMENTS

We would like to thank our shepherd Ripduman Sohan and the anonymous reviewers for their insightful feedback. This research was supported by the National Science Foundation under grants CNS-1050170, CNS-1017545 and CNS-0746531.

9. REFERENCES

[1] Ai enters the mainstream. `http://www.domain-b.com/infotech/itfeature/20070430_Intelligence.htm/`.
[2] Apple's iPhone 4: Thoroughly reviewed. `http://anandtech.com/show/3794/the-iphone-4-review/12`.
[3] Cmu sphinx. `http://cmusphinx.sourceforge.net`.
[4] Developing enterprise applications for mobile devices remains way too hard. `http://zdnet.com/blog/gardner`.
[5] Google android. `http://android.com`.
[6] Health information privacy. `http://hhs.gov/ocr/privacy`.
[7] Oracle virtualbox. `http://virtualbox.org`.
[8] Why your smartphone battery sucks. `http://pcworld.com/article/228189`.
[9] R. K. Balan, M. Satyanarayanan, S. Y. Park, and T. Okoshi. Tactics- based remote execution for mobile computing. In *MobiSys*, 2003.
[10] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: implications for network applications. In *IMC*, 2009.
[11] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: taking control of the enterprise. In *SIGCOMM*, 2007.

[12] B. Chun and P. Maniatis. Augmented smartphone applications through clone cloud execution. In *HotOS*, 2009.

[13] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. CloneCloud: elastic execution between mobile device and cloud. In *EuroSys*, 2011.

[14] B.-G. Chun and P. Maniatis. Dynamically partitioning applications between weak devices and clouds. In *MCS*, 2010.

[15] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making Smartphones Last Longer with Code Offload. In *MobiSys*, 2010.

[16] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *OSDI*, 2010.

[17] L. Fiering and K. Dulaney. iPads: Not notebook replacements, but still useful for business. *Gartner, Inc.*, 2010.

[18] J. Flinn, S. Park, and M. Satyanarayanan. Balancing performance, energy, and quality in pervasive computing. In *ICDCS*, 2002.

[19] A. Greenberg, G. Hjalmtysson, D. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4D approach to network control and management. *ACM SIGCOMM CCR*, 2005.

[20] K. Kumar and Y.-H. Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 99, 2010.

[21] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM CCR*, 2008.

[22] A. Messer, I. Greenberg, P. Bernadat, D. Milojicic, D. Chen, T. Giuli, and X. Gu. Towards a distributed platform for resource-constrained devices. In *ICDCS*, volume 22, 2002.

[23] S. D. Nelson and D. A. Willis. Separating enterprise tablet applications from consumer apps. *Gartner, Inc.*, 2011.

[24] A. Ramachandran, Y. Mundada, M. B. Tariq, and N. Feamster. Securing enterprise networks using traffic tainting. Technical Report GT-CS-09-15, GaTech, October 2009.

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

[26] S. Smaldone, B. Gilbert, N. Bila, L. Iftode, E. de Lara, and M. Satyanarayanan. Leveraging smart phones to reduce mobility footprints. In *MobiSys*, 2009.

[27] J. R. Tyler, D. M. Wilkinson, and B. A. Huberman. Email as spectroscopy: automated discovery of community structure within organizations. *Communities and technologies*, pages 81–96, 2003.

[28] T. Whalen, D. Smetters, and E. F. Churchill. User experiences with sharing and access control. In *CHI*, 2006.

[29] A. Wu, J. M. DiMicco, and D. R. Millen. Detecting professional versus personal closeness using an enterprise social network site. In *CHI*, 2010.

[30] H. Yan, D. A. Maltz, T. S. E. Ng, H. Gogineni, H. Zhang, and Z. Cai. Tesseract: A 4D network control plane. In *NSDI*, 2007.

[31] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, Z. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *CODES+ISSS*, 2010.

[32] X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham, and S. Jeong. Securing elastic applications on mobile devices for cloud computing. In *Workshop on Cloud computing security*, 2009.