An Architecture for Enhancing Capability and Energy Efficiency of Wireless Handheld Devices*

Rajesh Palit, Ajit Singh, Kshirasagar Naik
Department of Electrical and Computer Engineering
University of Waterloo, Ontario N2L 3G1, Canada.
{rpalit,asingh,snaik}@uwaterloo.ca

Abstract

With the tremendous growth in the hardware miniaturization technology, there are many types of digital electronic gadgets being used in daily life for different purposes. These devices are built to work alone and they typically cannot access or share each other's hardware or software resources. But the fact is that such devices are constrained in battery-energy and resources. In the presence of a generic resource sharing infrastructure, these are able to share and access each other's hardware, software resources, and data. Thus these devices become more efficient in terms of energy expense, and more enhanced in functionality, and usability. In this paper, we propose the concept of Universal Computing and Communication Interface (UCCI) that facilitates such sharing of resources between two wireless portable devices. This model comprises two basic components: Device and Connection Management (DCM) protocol and Framework for Information Exchange (FIX). DCM *devises a unique way to save energy by allowing a server to stay in* sleep *state while its service is not needed. On the other hand, FIX enables software applications on a small device to use resources such as CPU, Internet bandwidth and storage available on a larger computer. We have used state-of-the-art smartphones* HTC Nexus One, BlackBerry 9700 *and a laptop to develop prototypes of the proposed idea. We have conducted extensive experiments on the devices and measured the real-time energy consumptions. This papers explains situations under which such resource sharing can lead to energy saving. We also assessed the latency in accomplishing a task performed through sharing of resources.*

Keywords: Wireless networks, portable devices, energy efficiency, usability;

1 Introduction

In this section, we provide background and motivation for the work presented in this paper. Then, we describe the system model, design criteria and research objectives of this work which are followed by the summary of contributions. We discuss the organization of this paper at the end of this section.

* *A preliminary version of this paper was published in the proceedings of 22nd IEEE Symposium on Personal, Indoor, Mobile and Radio Communications (IEEE PIMRC'11).*

1.1 Background

In recent years, there has been a revolution in the industry of wireless portable devices. Smartphones are equipped with essential gadgets such as global positioning system (GPS), digital camera, and multiple communication interfaces. As a result, the functionality of these devices is not limited to exchanging voice calls, rather users use their phones to access email, browse Internet, and play multimedia contents. The features and functionalities of these devices are improving day by day with reduced size and price. Accordingly, the usage of these devices are becoming more and more common in daily life and user expectations in terms of running heavier applications are rising rapidly.

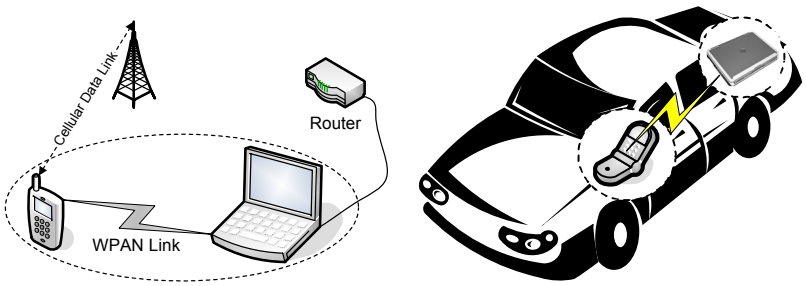
These devices are generally powered by small, re-chargeable batteries, and unfortunately, the growth in battery technology has not kept pace with the rapidly growing energy demand of these smart devices [1]. For example, the battery of a state-of-the-art smartphone lasts only $3-4$ hours when online video is played. A *GPS* aided navigation application runs around the same amount of time when it runs solely on battery. This dependence on battery energy puts a severe constraint on the availability of these devices [2]. Moreover, it is not feasible to equip the handheld devices with full-featured hardware components due to size and limited battery energy. Consequently, these devices are not capable of running resource intensive applications, which limits the functionality of these devices. In comparison to the size and weight of the smartphone, laptop computers are large and heavy, with relatively higher capacity CPU, battery, and communication bandwidth.

1.2 Motivation

Due to the complementary attributes of laptops computers and smartphones, nowadays professionals, business executives and university students use laptops as well as smartphones to meet their computing and communication needs. However, they often feel the necessity for sharing resources between these devices. For example, when they work on their laptop, they like to access some files, 3G data networks, or even the on-board camera of their smartphone; or, they may need to access the high capacity processor, high bandwidth data network available on the laptop while they work on their smartphone. Sometimes they need to access a licensed software from one device to another. Most importantly, they often need to share their data among these devices.

In Fig. 1(a), a smartphone user is connected to Internet through a laptop. The user might be interested to do it if the phone is not subscribed to cellular data services, or even with a subscription to data service, the user may still divert its Internet traffic through the laptop. Because the data service may not be available in some places or, the connection speed might be poor, or the cost of the data service is high. Fig. 1(b) shows a scenario where a smartphone user is retrieving a file from his/her laptop as the laptop is not accessible for the time being. This may happen when the laptop is in the trunk of a car or in the airplane overhead cabin box, or, the laptop may run short of battery energy, so the user does not want to open it. The display on the laptops consumes around 36% of its total energy consumption [3], so it could be a simple case of energy awareness.

Thus the shortcomings of resource constraints in portable devices can be overcome by sharing resources which results in functionality enhancement. In addition to that a device is able to save its battery energy by using resources of other devices instead of its own. In fact, energy saving was the most significant factor for offloading tasks from a mobile device



(a) A smartphone is accessing Internet through a computer.

(b) A user is retrieving a file from laptop.

Figure 1. Smartphone is communicating with a laptop in two different scenarios.

to a server device [4, 5, 6, 7, 8]. The benefits sharing resources are threefold: (i) a device acquires some functionalities which it does not possess itself; (ii) it is able to access some resources which does not belong to it; and (iii) it saves energy. All of these benefits are very much appealing to the users, and the devices become more useful to them.

1.3 System Model and Design Criteria

As shown in Fig. 1(a), the participating devices can communicate with each other via multiple communication links such as wireless local and/or personal area networks. We refer a device as a server when it allows other devices (clients) to access its resources. A server can be a laptop, a desktop computer or some other dedicated device having computation and communication facilities. We assume that the server device permits the clients to do so because they belong to same owner or the server device gets incentives for rendering services. In the following, we describe the design objectives to accomplish the task of resource sharing in energy efficient manner.

- *Communication Link:* Connection establishment is a prerequisite for sharing resources among participating devices, and Internet can be used to establish such connections. However, Internet is not available when a user remains outside of user's workplace, and also, integrity and security of the personal data always remain a major concern. Also, routing data through Internet where source and destination are in close proximity will unnecessarily burden the Internet. Therefore, a local communication link that requires low energy, and meet security constraints is suitable for this purpose. On the other hand, a wired connection requires a physical contact by cables, and it also limits the movement of a device, thus a wireless link is much preferred for this purpose.
- *Service On-demand:* A server is able to provide instant access to its resources when it remains available all the times. Consequently, it consumes energy to remain always *On* (further discussed in Section 3). Thus when a server is not used frequently, it wastes energy most of the times just to remain available. This situation can be avoided if a server becomes available only when its service is needed. We call it *service on-demand*. A client device awakes a server when it needs service. Here, saving of energy comes with a latency in getting service after a request is made.
- *Software Framework:* In addition to the communication link between client and sever, there must be an agreement among them so that they are able to communicate, and transfer data for sharing resources. There are diverse types of devices available in

the market, and they come with different platforms (*operating systems*). Therefore, a generic framework (*cross-platform*) is essential to accomplish the task of resource sharing.

- *Energy Saving:* Sometimes a client accesses resources on a server to attain some functionality which that client does not possess, and in such scenario, energy saving is not an objective for both client and server. However, in some situations, a client accesses high capacity resources of a server to save its time and energy. In such cases, we assume that the server comes with an adequate supply of energy, resources. and we mainly focus on energy saving in the client device. The condition of saving energy is relaxed only when functionality enhancement is prime objective.

1.4 Research Objectives

We propose a generic architecture called *Universal Computation and Communication Interface (UCCI)* to enable communication between a mobile client and a nearby server device. Prior work advocates only for offloading a computational task to another device, but we have included the communication and sharing of resources in our model. We also consider the energy expense of the server device, because in our model the server itself can be a portable device running on battery, and the server device can remain in *sleep* state. *UCCI* consists of *Device and Connection Management (DCM)* and *Framework for Information Exchange (FIX)* protocols. *DCM* uses personal area wireless link (Bluetooth) to communicate a server, and it uses the 'Wake ON' feature of the server to *awake* it from *sleep* state to *active* state. *DCM* puts a server again in *sleep* state when service is not needed. *FIX* works on top of *DCM* and it facilitates the sharing of information and resources between a client and a server.

'Wake ON' feature allows a device to be turned on by a network message. Such a device can be kept in a very low power state by turning on the 'Wake ON' feature, and make the device available for use when necessary. We have conducted a set of experiments to observe the energy costs of a devices in different power consumption states with and without turning on the 'Wake ON' feature. Results show that the energy costs (overhead) for activating 'Wake ON' feature is very less.

We need to know the steps that take place in *UCCI* connection to evaluate the impact of resource sharing on energy consumptions. The data processing rate and energy costs of basic operations such as communication, computation, storing and retrieving data from storage need to be investigated to estimate the costs of performing a task on a device. Then, we are able to compare the costs with and without resource sharing. Suppose that a smartphone, x accesses a resource, r (CPU) on a laptop, y. To accomplish this, x needs to follow the steps [5]: (*i*) needs to maintain a connection with y, (*ii*) reads data and related codes from its storage, (*iii*) sends them to y, (*iv*) waits for the results, and (*v*) receives the results. Data speed of the communication link between the devices and the data processing rate at device y contribute to the latency of the overall process, and the energy spent in transmitting and receiving data constitutes the energy cost. The reading and writing of data from and to the storage remain the same when a task is accomplished in device x. There is also some costs incurred in x for staying active while the task is processed in y, however, it can be avoided by properly scheduling the device x to awake up when the task is completed. We measured these costs on a state-of-art smartphone, *HTC Nexus One*,

and discuss the possible scenarios when and how sharing of resources can be effective and efficient.

1.5 Summary of Contributions

The main strength of this work is that we not only have designed a model, we have also developed prototypes to show the validity of the model. We have implemented the system using the off-the-shelf devices. We summarize the principle contributions of our work below.

- (c_1) We introduce the concept of *UCCI*, a generic model for sharing resources among portable wireless devices. *UCCI* consists of two protocols *DCM* and *FIX* that enable any device communicate with another device without having prior knowledge of each other. when a device finds that it does not have the functionality or enough resource to execute a task or it intends to save energy, the device exports that task to a nearby server. when a device has multiple communication links available to it, the device can pick a suitable one to fit its need and for saving energy.
- (c_2) Two prototypes have been developed on Android and BlackBerry smartphones namely, *HTC Nexus One* and *BlackBerry 9700* to demonstrate the efficacy of the model.
- (c_3) To observe the energy saving aspects of *UCCI*, we conducted experiments to measure the application level energy and communication costs for different usage scenarios.
- (c_4) We discuss the experimental results and explain how and in what situations resource sharing can be effective, and save energy with less delay. We also discuss the various security aspects, and argue that the model does not give rise to any new security threats.

The rest of this paper is organized as follows. In section 2, we discuss related work such as task offloading and low power personal area network communications. In section 3, we explain the working principle of our framework. Section 4 describes the prototype implementation of the proposed framework and it is followed by experimental setup in section 5. Section 6 presents the experimental results with discussions. We put conclusions in section 7.

2 Related Work

We review some substantial prior work in this section. We begin with discussing the benefits and costs of offloading followed by the different techniques or methods of offloading. We also explain the reasons behind different design issues of our model. We finish this section with a discussion of the reasons for considering Bluetooth as the client to server device communication link.

Gitzenis *et al.* [2] studied the problem of task offloading and power management in wireless computing. They presented a Markovian dynamic control framework to optimize the task migration and processor speed/power management. They pointed out that task offloading results into energy savings at the mobile terminal (sparing its processor from computations) and execution speed gains due to (typically) faster server processor(s). However, the overheads are the energy cost for terminal-server wireless communication and the delay

for uploading the task and getting back the results. The net gains (or losses) depend on network connectivity and server load. Their observation is: for a task with low communication and high computation requirements, migration is advantageous under both criteria of energy consumption and response delay. However, to accomplish this, the wireless connectivity must be strong and the server has to be lightly loaded. Weak connectivity turns migration into a less attractive option. In our model, we consider the short range personal area wireless link which is robust with relatively moderate bandwidth. For example, Bluetooth v2.1 supports 2 Mbps application to application data rate and Bluetooth v3.0 supports up to 24 Mbps. According to the observations in this research, our proposed model and its operating environment are suitable for task offloading.

Xhao *et al.* [9] studied a case where resource limitation forces offloading of a task. They propose a H.264 encoder modularization and energy models for offloading. They mainly focus on the usage of the computation offloading method to H.264 video encoder on mobile devices. Results from three of their offloading schemes show that offloading the encoding part of inter frames or the whole video encoder can save large amounts of energy. They observe that with efficient wireless link, computation offloading techniques would be more efficient to save energy on mobile devices.

Rudenko *et al.* [10] present an automation framework for computation offloading and it records the average power consumption of a repetitive task for deciding whether to offload the task. Kremer et al. [11] propose an offloading scheme that uses check-pointing techniques to handle disconnection events for wireless connection. The cost model for local computation is based on the average computation time. Rong *et al.* [12] study offloading under real-time constraints. They use multiple synthetic tasks and each task has a known constant computation time. Li *et al.* [4] propose making offloading decisions at function level. The computation of each function is assumed to be a constant and obtained by profiling. Wang *et al.* [13] propose a method of parametric compiler analysis to determine the computation time. The method considers only simple parameters, such as the command line options. It cannot analyze more complex data such as an image. All these methods require estimating the computation time before execution in order to make offloading decisions. In contrast, Xian *et al.* [6] use a timeout method and do not require such estimation. In their study, a timeout is set for computation instead of an idle period. If the computation is longer than the timeout, the computation is offloaded to a remote server to conserve the energy for the client.

Gurun *et al.* [5] present a framework for making computation offloading decisions in computational grid settings in which schedulers determine when to move parts of a computation to more capable resources to improve performance. They argue that offloading decision amounts to predicting the bandwidth between the local and remote systems to estimate costs associated with offloading. They authors compared a classical approach to several variations of a Bayes decision model and a no data approach. And found that a Bayesian approach which incorporates change-point detection in its formulation of the prior distribution is the most efficacious of those they investigated.

Gu *et al.* [7] propose an adaptive offloading system that includes two key parts: a distributed offloading platform and an offloading inference engine. They mainly focus on the memory. And, when the application memory requirement approaches the mobile device's maximum memory capacity, the system initiates offloading. The system partitions the application's program objects into two groups, offloading some to a powerful nearby surrogate to reduce the device's memory requirement. With the offloading inference engine, runtime

offloading can effectively relieve memory constraints for mobile devices. Having studied the pros and cons of the methods of offloading, we designed a very light and simple Offloading Decision Maker (ODM) engine. At first, it does not partition a task running parts in the mobile device and parts in the server. Rather it decides before executing a task whether to offload or not. In case of offloading, the whole task is migrated to server and an interface is executed for sending data and receiving the results. In making the decision, *ODM* considers the applications resources requirements, availability of offloading service, system's resource meter and energy state and above all user's permission.

Mahmud *et al.* [14] emphasize on having an energy-efficient scheme for simultaneous or single operation of the wireless interfaces attached to Multi-service User Terminal (MUT). MUT stands for the devices that have multiple wireless interfaces for receiving various classes of services from the networks. They propose a simple model for predicting energy consumption in a terminal attributed to the wireless network interfaces. Then, the actual consumption patterns are measured to estimate the parameters of the model. They observe that each access technology has a different data rate, network latency, interaction capability, mobility support, and cost per bit because each has been designed with specific services in mind. They stress on the need to have comprehensive understanding of the power consumption of the devices/modules in various operational states. Complying with this understanding we explored the energy cost of communication on different communication links that a smart phone typically possesses. And we use the results in designing our system model.

In mobile to server device communication we use Bluetooth link. Because, Bluetooth is widely adopted as short range communication protocol and almost all the smart phones come with a Bluetooth connectivity. More importantly the next generation Bluetooth v3.0 [15] module is going to be more energy efficient, more secure and is supposed to provide higher data rates of around 24Mbps. We summarize the strength of this work below:

- The concept of awaking a server *on demand* basis is very crucial. Instead of spending energy by keeping the server awake all the times, the server device saves significant portion of energy by being in *sleep* mode. We may compare the situation with constant polling versus interruption when the an event occurs.
- The idea of task exporting has been around for quite sometime and the design objective or target platforms were assumed to be grid or mesh networks. We view task offloading between peer devices, where functionality enhancement is the key objective and energy saving comes as a by product.

We have not only proposed a design or concept, we have implemented the whole system with existing hardware available in the market. This is the most important strength of this work.

3 Architecture

In our model, a device is termed as a client or a server based on its role or functionality in an ongoing session. A client device in a session may act as a server during some other session when another device seeks a service from it. So the terms client and server are not tightly coupled with a device. For example, when a smartphone accesses resources

of a laptop, the laptop becomes a server and the smartphone becomes a client. On the other hand, the laptop becomes a client when it accesses the files of the same smartphone. In this case, the smartphone acts as a server. The working principles of the three *UCCI* components are given below.

3.1 Device and Connection Management (*DCM*)

A server device can stay in several states of operation as shown in Fig. 2 and it is able to provide service in *active* state. In *sleep* (or *inactive*) state, it suspends all of its operations except the 'Wake On' feature. 'Wake On' feature refers to the capability of waking up from *sleep* state to *active* state after getting a special message through a communication interface such as LAN, Wireless LAN (WLAN) and Universal Serial Bus (USB). To enable 'Wake On' feature, a device needs to scan for particular message while sleeping. When both client and server stay in the same network, a client device needs to know the Internet protocol (IP) address of the server device. However, a client needs to know the Medium Access Control (MAC) address of the server to 'Wake Up' the server by sending *magic* packet.[1]

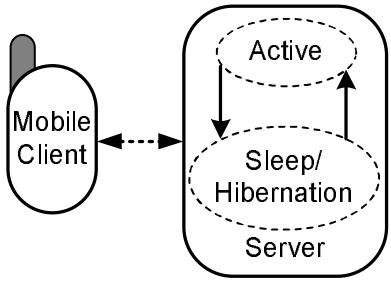


Figure 2. State Transition diagram of server device.

Now, let us explain how a client obtains the address of a server device:

- Server in *active* state: An *active* server periodically broadcasts its service list with its device address. Before initiating a connection, a client scans for devices surrounding it and makes a list of devices. The user of the client is then asked to select the server from the server list.
- Server in *sleep* state: When a server is in *sleep* state, it does not broadcast its address, and in this case a client needs to get the address from the user or from history data. If the server device belongs to the same owner, he/she knows the device address and for a public server, the address of the device needs to be printed on it. Otherwise, a client won't be able to wake it up. Usually, a client device keeps a short list of devices which are connected quite often, and thus a user does not require to input the server address all the times. As all the devices have user-friendly names, users are not expected to input bizarre hexadecimal device addresses.

A client does not need to send a WAKEUP message to turn on the server as it is already in *active* state. However, client sends a HELLO message to check whether the server is

[1]A magic packet is sent to *wake up* a device from sleep state. This packet contains 255 in consecutive 6 bytes, followed by 16 repetitions of the target device's 6-byte MAC address anywhere within its payload. The magic packet is typically sent as a UDP datagram to port 7 or 9.

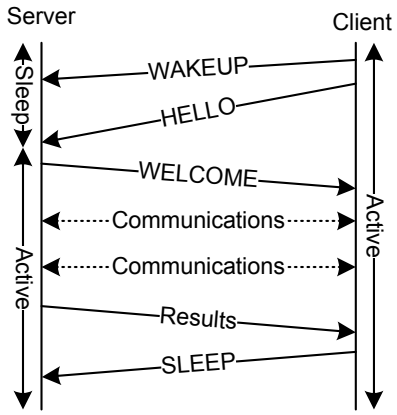


Figure 3. Timing diagram of task offloading using UCCI.

ready to accept a connection. For an *inactive* server, a client sends a WAKEUP message to activate the server. The relative timing of the events is given in Fig. 3. After sending the WAKEUP message, the client waits for a while (10 seconds) for the server to come in *active* state. It then sends a HELLO message to check whether the server becomes active. The server replies with a WELCOME message. If the client does not receive a WELCOME message within a time frame (5 sec), it again sends HELLO message. In the HELLO message, the client mentions its identity and service request. Thus the server transfers the controls to specific handlers. After completing the task, the client sends a SLEEP message to put the server in sleep mode.

3.1.1 Hardware and Software Requirements

The required hardware and software tools necessary to implement the *UCCI* model are already available in the marketplace and we now describe hardware and software requirements for server and client devices below:

- Hardware requirements for server: A server device is equipped with a low power short range high speed wireless communication module to communicate with the client. The server also has 'Wake On' feature so that it can be awaken from *sleep* state. Otherwise, a server needs to be ON all the times to make its services available, and for a portable device it is not affordable.
- Hardware requirements for client: A client device includes the same low power wireless communication module to connect with the server. It is capable of sending special 'Wake UP' signal using the device address of the server so that the communication module at the server can wake up the server from *sleep* mode. The client's module also capable of searching surrounding server modules and services.
- Software requirements for server/client: After establishing the connection between server and client, the server verifies identity of the client, and based on the requested service type, the server transfers the handle to the specific handler. In our implementation, we used Java and no specialized tools was necessary to develop the applications on the client and server sides.

Hardware components with energy saving features [16] are available in the digital system since long. Personal computers are equipped with 'Wake On LAN', 'Wake On USB' features.

Some of the Bluetooth and WLAN hardware components have 'Wake On' capability and some Operating Systems (OS) such as Mac OS supports these features. However, 'Wake On Bluetooth' can easily be incorporated on the OS if hardware supports this feature. Another way to enable 'Wake On' option is by exploiting 'Wake On USB' feature. It is done by attaching an USB Bluetooth module in the device's USB port.

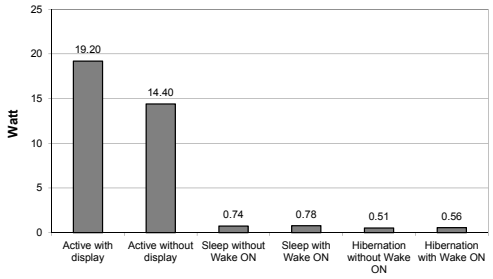


Figure 4. Power consumption of a laptop in different states.

3.1.2 Energy Saving Measures

The graph in Fig. 4 depicts the benefits of *sleep/Wake-UP* model of the server. A laptop that we have used in our experiment consumes 19.2 and 14.4 watts in *active* state, with and without display, respectively. Whereas, in *sleep* and *hibernation* states, the laptop consumes only 0.78 and 0.56 watts, which are just 4.06% and 2.92% of the *active* state, respectively.

Fig. 4 also shows that the increased energy consumption for turning on the 'Wake On' feature is only 5.4% and 10% more in *sleep* and *hibernation* states respectively. However, 'Wake On' feature is needed to put an *active* server to *sleep* state and put it back to *active* state when it is requested for service. As a result, it is definitely energy saving to adopt the *sleep/Wake-UP* model instead of being always *active*. It is worth to mention that a device takes time to switch from *hibernation* to *active* or *sleep* to *active* states. These delays usually range from 5 to 20 seconds and they vary system to system. The delay for switching from *hibernation* to *active* state is longer, and the decision whether to put a server in *hibernation* or *sleep* state depends on the type of application. For quicker response time, a client needs to put the server in *sleep* state rather than in *hibernation* state.

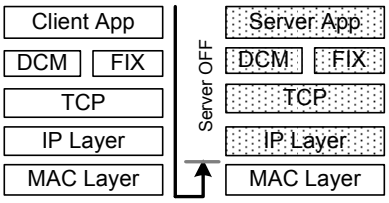


Figure 5. Placement of UCCI in protocol stack.

3.1.3 Placement of UCCI in OSI Model

Fig. 5 shows the position of *UCCI* in the network protocol stack. When a server device is in *sleep* state, a client can only reach its MAC layer by sending a WAKEUP message.

After getting the WAKEUP message, server switches to *active* state and the client is able to access the server application, and further interactions take place.

3.2 Framework for Information Exchange (FIX)

The software applications such as remote file browsing, sharing, remote desktop sharing, Internet sharing, sharing camera or *GPS* data basically involve exchange of data. They are productivity or utility tools which are simple yet they provide the users of these devices with very useful services. One practical example would be very relevant here, users tend to take weeks to download the files of captured videos and photos from digital cameras. Because they need to connect devices to computers via cables. In the following, we describe how *UCCI* facilitates these types of tasks using *FIX*.

When a client connects a server device, it initially retrieves service information from the server. Then, the client device lays out the service list, and allows its users to choose an option as shown in Fig. 6. When a user selects an option, the client device sends the corresponding code to the server, and server execute corresponding module to further communicate with the client device.

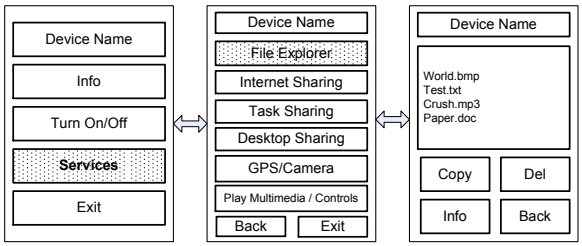


Figure 6. User interface of an UCCI based application.

Some software applications require extensive computation power and large memory capacity and claim good amount of energy. Such types of applications can be outsourced to take the advantage of CPU and memory on server devices. As we discussed in section 2, many techniques have been proposed in the literature regarding task offloading. For task offloading, the application need to have options for local and remote processing and when user select remote processing, the application transfer and manage through application programming interface (API). In our prototype, we have used Java to implement task outsourcing feature.

3.3 Possible Security Issues

We point out the possible security threats that are involved in *UCCI* model, and discuss below how these issues can be resolved by taking appropriate measures.

- Authentication: The UCCI service should be restricted to the authorized users and devices. In our model, we consider Bluetooth which has built-in authorization module using PIN codes. Therefore, no user can connect to another device without knowing the PIN code.
- Secure communication link: As the data and tasks are migrated from one device to another, we need to ensure that the communication link is secure in the first

place. Likely, secure personal area wireless communication is available and Bluetooth v3.0 supports 128-bit Advanced Encryption Standard (AES) which is state-of-the-art security protocol [17]. However, encryption and decryption of data consumes much energy and its use is recommended only when there is a need.

- Software Security: When a task is transferred to a server, we need to ensure that it does not break server's security, and also no other application on server can intercept the data or code of the task. In UCCI, an exported task is executed on a Java Virtual Machine (JVM), and therefore, in the controlled environment, the task cannot break the security of the server, or any other application can access the information of others.
- Hardware Security: In our model, the server or client devices are typically belonged to the same owner, so the threat of hardware intimidation is less in this case. Hardware intimidation is not a new security threat arises from our model, and users need to be cautious about this when they connect to a public device.

4 Prototype Implementation and Model Validation

To implement prototypes for the proposed *UCCI*, we have used a *Toshiba Tecra R10-ES1* laptop as a server device with Windows 7 operating system on it. A *HTC Nexus One* and a *BlackBerry 9700* smartphones were used as clients to build our prototypes. The smartphones communicate with the laptop through its built-in Bluetooth link. However, the operating system does not allow the Bluetooth device to *wake* it up from *sleep* state. We used an external mouse (*Microsoft Wireless Laser Mouse 8000*) to facilitate that. An USB dangle is attached with laptop and the mouse communicates with the dangle using Bluetooth link. The USB dangle connects the laptop as human interface device (HID), and thus the mouse is able to *awake* the laptop from *sleep* state. If the OS supports the 'Wake On' feature of the built-in Bluetooth device, the external mouse would not be needed. With an appropriate device driver, features of these two Bluetooth devices can undoubtedly be combined. However, we avoided that as we have developed only the prototype of our concept.

Figure 7. Snapshot of an Android based UCCI application.

We developed an application for server and two versions of the client application to implement *UCCI*. All applications are written in Java, and the client applications are

modified to fit with Android and BlackBerry OS. A screenshot of the client application is given in Fig. 7. It shows Android OS version of the application which is running on the *HTC Nexus One* smartphone. The user interface of the BlackBerry version is the same and is installed on a *BlackBerry 9700* device. The applications are able to establish *UCCI* connections via both the WLAN and Bluetooth links. The 'Wake On' feature of the WLAN works only in presence of AP, and it is not supported in WiFi adhoc mode. Moreover, WLAN link becomes useless while the user on the road or in the place where there is no support 'Wake On' packet forwarding.

Using the application shown in Fig. 7, the smartphone (HTC Nexus One) wakes up the server (Toshiba laptop) from *sleep* state, and transfers data using Bluetooth link. After processing the data, the results are sent back to the smartphone, and the laptop is put back into *sleep* state again. In fact, once the laptop is connected with the smartphone, we can access and use the resources on it, it is just a matter of attaching appropriate software applications. We play audio/video, and control the volume on the laptop from the smartphone, and similar activities can be performed on the smartphone from a laptop.

5 Experimental Setup

As discussed in section 1, we need to know the energy costs of the basic operations such as computation, communication and storing data on a device, so that we are able to estimate the energy saving when sharing of resources takes place. If a device spends much energy in transferring data to the server, the net energy gain becomes less, in fact, it can be negative in some situations. In this section, we describe the experimental setup that we used to conduct experiments for evaluating energy costs.

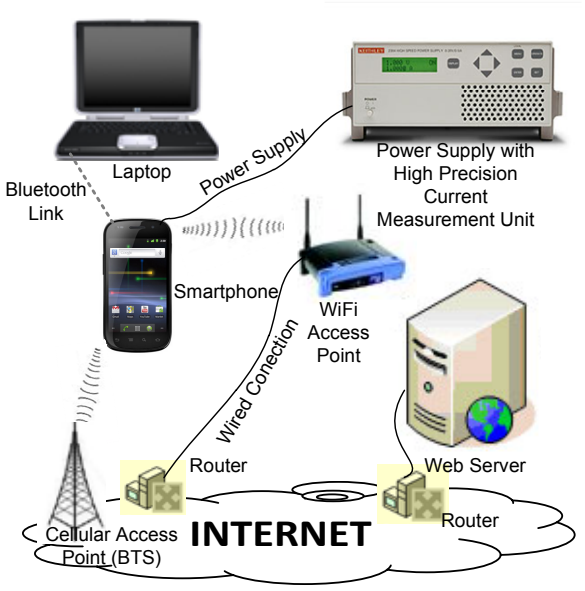


Figure 8. Logical view of our experimental setup.

As shown in Fig. 8, we connect the *HTC Nexus One* smartphone with a Keithley 2304A high speed power supply to measure the current consumption of the smartphones. The power supply is connected to a desktop PC via USB port, and using a controller program

on that PC, we set required output voltage (3.7 volts) on the power supply. We sampled the consumed current by the smartphones one second interval and readings were taken throughout the execution of an operation. For example, before initiating a file transfer using SFTP (Secure File Transfer Protocol) we start probing current, and stop after the transfer stops. We repeat the experiments at least 3 times, and each time we take 5 sets of readings for each scenario. Each set contains 50 to 150 readings based on the duration of the operations. We show the variations in the readings by providing the maximum, minimum and mean values of the readings. Prior to conducting experiments, we configured the settings and battery connections of the smartphone as described in [18].

6 Results and Discussions

In this section, we present the energy costs of some basic operations such as computation, communication and data storage and retrieval for *HTC Nexus One*. Then we show the costs of transferring a file from the same smartphone to a server located in the Internet.

6.1 Energy Costs for Basic Operations

We chose three widely used CPU benchmark programs, namely, *Linpack*, *Whetstone* and *Dhrystone* to evaluate the cost of computation on *HTC Nexus One* smartphone. We used another Android application which computes the exponents of two random real numbers continuously. We executed these four applications for 20 seconds each time, and measure the current consumptions. Fig. 9 shows the power consumptions of the smartphone for each of the applications and, we see that the benchmark programs consume about 961 milliwatts. On the other hand, the exponent computation application consumes about 1100 milliwatts as it involves only floating-point operations. Once we know the energy cost of CPU for full utilization, the energy cost of an application for computation can be calculated based on its CPU utilization [16].

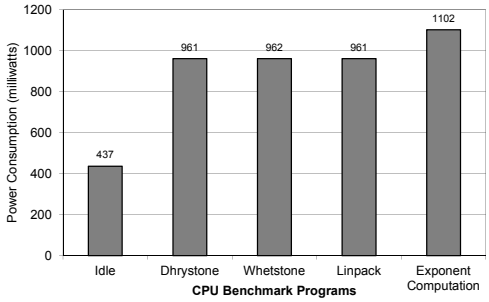


Figure 9. Power consumptions for computation.

We obtained the energy costs of reading and writing in the built-in storage of the same smartphone by a simple application which reads and writes data in blocks of different sizes. We varied the block size from 256 bytes to 8 kilobytes, and the application reads a file of 256 megabytes, and write a file of 128 megabytes with random contents. The reading process takes around 30.5 seconds, and writing takes around 36 seconds irrespective of block size. The results in Fig. 10 show that smaller block size consumes more power.

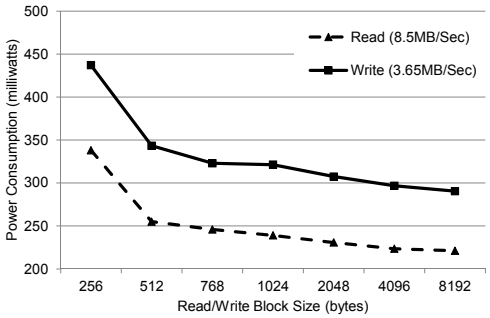


Figure 10. Power consumptions for reading and writing data in the internal storage.

Fig. 11 shows the power consumptions for transmitting UDP data packets of 1472 bytes via WiFi interface (802.11*g*). The experiment was done on the same *HTC Nexus One* smartphone. The data rates were varied as we changed the transmission interval between two packets. It is interesting to observe that for smaller transmission intervals the power consumptions are same. The maximum size of an unsegmented UDP packet that can be sent from application layer is 1472 bytes in this scenario. More information on the impact of packet size and delay on power consumption can be found in [19]. Though we have shown only for WiFi interface, the effect of packet length and intervals is also significant in other interfaces.

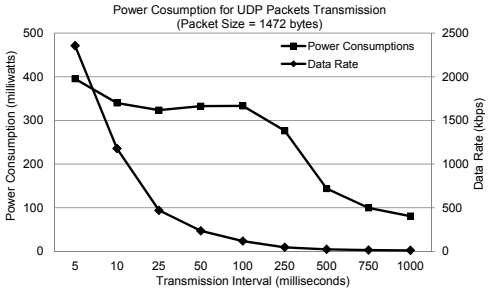
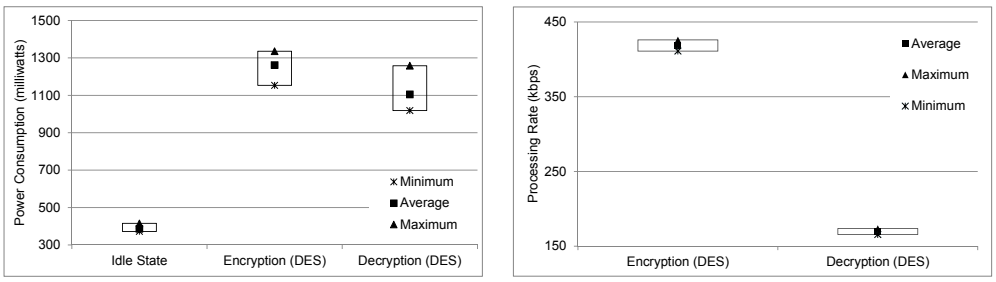


Figure 11. Power consumptions for transmitting data packets via WiFi (*802.11g*).

We also measured the cost of data encryption and decryption with an application that uses Java standard API for 64-bit DES (Data Encryption Standard) encryption and decryption algorithm. The application reads a large file of 32 megabytes and writes the same file after encryption. Later on, the encrypted file is read back by the application, and stored as a separate file after decryption. The power consumption and processing speeds of the processes are given in Fig. 12. We see that though the power consumption for decryption is less than the encryption process, the data processing speed for decryption is significantly less in compare to the encryption process. We see the impact of this during secure file transfer process that we discuss later in this section.

With the help of energy cost information presented above, we are now in a position to make energy-aware decision. For example, it requires 1233 Joules of energy to store in the internal storage, and another 412 Joule to read the file when necessary. If we want to store the file in a storage server in the Internet, with basic cost information, we can calculate the required energy for transmitting and receiving the file over the communication with appropriate security measures. Of course, storing in the network server involves more



(a) Power consumption for encryption/decryption.

(b) Obtained data rate for encryption/decryption.

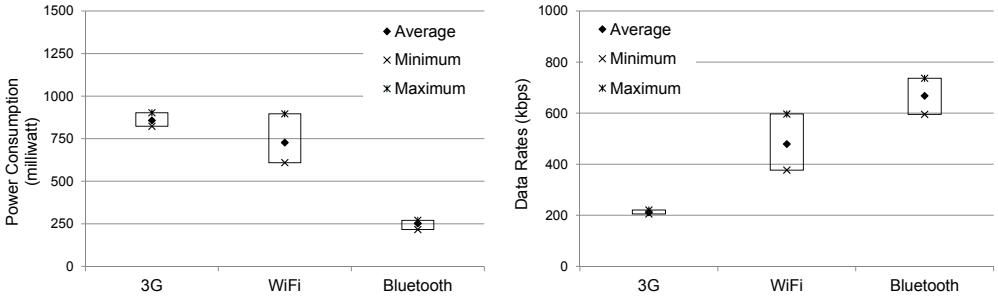
Figure 12. Power consumptions and obtained data rates for encrypting and decrypting data (64-bit DES with 2 KB block size).

energy, specially when we need to access the file frequently. However, when the device is running short of storage space, we compel to store it on the network server.

6.2 Energy Costs for Transferring a File

Now we investigate the energy consumptions of a task which involves resource sharing via *UCCI*. We choose to transfer a file from the smartphone to a SFTP server located in the Internet. This task of transferring a file from the smartphone can be accomplished in the following ways:

- **Scenario 1 (S1):** send the file from smartphone directly via 3G link,
- **Scenario 2 (S2):** send the file to the laptop via Bluetooth link, and then send it to the SFTP server from the laptop,
- **Scenario 3 (S3):** compress the file in the smartphone, and then send the compressed file via 3G link, and
- **Scenario 4 (S4):** send the uncompressed file to the laptop, compress it there, receive it from the laptop via Bluetooth link, and send the compress file via 3G link.



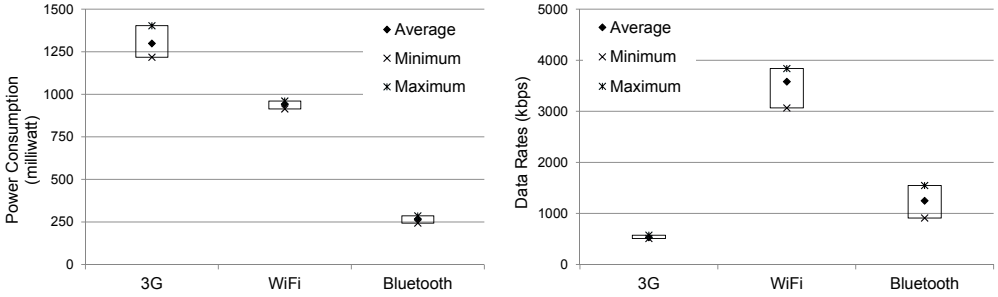
(a) Power consumption for downloading.

(b) Obtained data rate for downloading.

Figure 13. Power consumptions and obtained data rates for downloading data via different links.

We measured the energy consumed by the smartphone in each of the above scenarios. For that, we, at first, evaluated the power consumptions and data rates for 3G (WWAN) and Bluetooth (WPAN) links to estimate the energy costs for transferring data. We also

measured the cost of transferring data over WiFi link as it falls in between WWAN and WPAN in terms of coverage. We conducted the experiments in different time of the day, and repeated the experiments to avoid any temporary fluctuations in the measurements. It is worth to mention that these results are dependent on the location of the mobile tower, WiFi access point and local interference, but we took no measures that might affect the data rates of any of these links. Therefore, we may consider these results as an instance of a typical scenario experienced by users.



(a) Power consumption for uploading. (b) Obtained data rate for uploading.

Figure 14. Power consumptions and obtained data rates for uploading data via different links.

Fig. 13 and Fig. 14 show values of the mean, minimum and maximum power consumptions and obtained data rates during downloading and uploading of files from/to the SFTP server. We observe that the data rates in uploading are higher than that of downloading, which is counter intuitive. We performed some additional experiments to verify this result. We used simple FTP protocol to transfer files, and found that data exchange rates are much higher in this case, and download rates are higher than upload rates. We also examined the energy costs of reading and writing data from/to storage. We found that writing data on *sdcard* consumes 35% more power than reading data whereas writing is done at 42% of the reading speed. Actually, when we send a file through SFTP, the following operations take place: reading the file from storage, encryption the file data, transmission of the data. On the other hand, when we receive a file via SFTP - reception of file data, decryption of the received data, and writing the data on the storage take place. In SFTP process, operations other than the transmission/reception speeds dominate the overall data processing rates. Further study is necessary to investigate this matter. Data are more vulnerable in the WWAN or WLAN than in WPAN and for that reason we used SFTP to transfer data. Fig. 15 shows the energy expense for downloading and uploading a one megabyte file. We see that 3G link costs 10 times more than the Bluetooth link when we transfer the file. Data transfer costs over WiFi link is also lower than that of 3G link.

To measure the energy costs for scenario S3 and S4, we measured the energy cost and data processing rates of compression and decompression process on the *HTC Nexus One*. As shown in Fig. 16(a), power consumptions for compression and decompression are 1236 and 1184 milliwatts, respectively. The data processing rates during compression and decompression are about 2456 and 4818 kilobytes, respectively (shown in Fig. 16(b)). Using the results from the experiment, we are now able to calculate energy costs of transferring files via 3G, WiFi and Bluetooth links with or without compressing the data. Compression ratio is the most important factor in deciding whether to compress before transferring

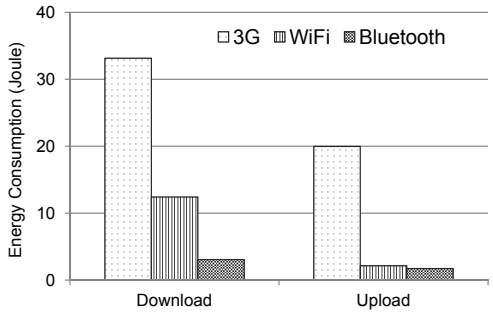
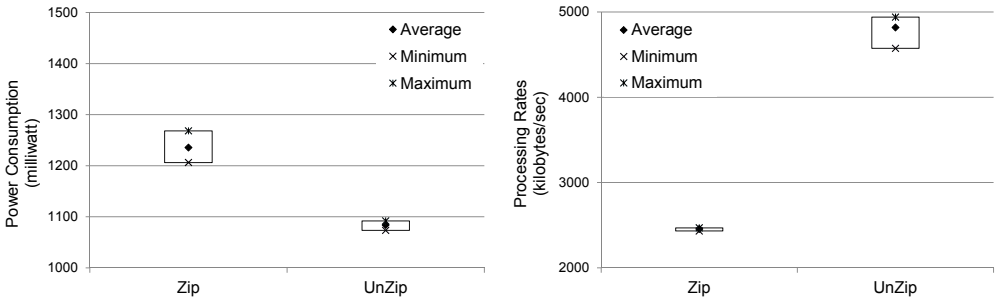


Figure 15. Energy consumption for transferring a file of 1 MB size.

data. The compression ratios of JPEG (Joint Photographic Experts Group) and MPEG (Moving Picture Experts Group) format files (users typically encounter in smartphone) are very low and sometimes become negative even after using efficient compression algorithms [20]. In our experiment, we used Android's *ZipInputStream* and *ZipOutputStream* class for compressing *portable data format* (pdf) files and the compression ratio was only 12.5%.



(a) Power consumption (milliwatt). (b) Processing rate (kilobytes).

Figure 16. Power consumption and processing rate for compression/decompression.

Table 1. Comparison of required time and energy costs for different Scenarios

Scenarios	Time Required (Second)	Energy Consumptions (Joule)
S1	76.99	99.93
S2	32.79	8.76
S3	69.45	90.02
S4	153.81	109.72

We calculated the energy costs and processing times for uploading a file of size 5 (five) megabytes using four scenarios mentioned in the beginning of this section. The results are given in Table.1. In scenario S2, the consumed energy is the least among the four scenarios. We assume that server has much higher Internet bandwidth, and we did not include the time to upload the file from *UCCI* server to SFTP server in scenario S2. In scenario S3, the cost is less than that of S1, but the difference is not significant. In fact, the compression ratio is

a critical factor here, higher compression ratio yields better energy saving. The completion time of whole uploading process is also important. Presented energy costs are the costs uploading process only. During the file uploading, the device needs to be active which consumes additional energy (about 450 milliwatts on our smartphone). Therefore, energy cost in scenario, S4 is the worst in all respects. The data rates and energy consumptions for WiFi link are much better than that of 3G link, and therefore, a secure WiFi link (if available) should be preferred over 3G link.

7 Conclusions

We propose *UCCI*, a generic architecture for facilitating communication between two wireless portable devices such as a smartphone and a laptop. This framework allows a server device to remain in *sleep* state unless its service is needed by a client device. It exploits the 'Wake On' feature of the server device and uses the personal area low power Bluetooth link, and then, optionally, puts the server back into sleep state. To validate our model, we have developed two prototypes using state-of-the-art *BlackBerry 9700* and *HTC Nexus One* smartphones. We discuss the impact of this model by measuring power consumption on the client in different states through on-device experimentation. We compared the costs of transferring a file over 3G, WiFi, and Bluetooth links and measured the energy costs of data compression and decompression processes to show how they affect the energy budget of a smartphones. We aim to design and implement a generic framework in Java to incorporate this model.

References

[1] R. Powers, "Batteries of low electronics," *Proceedings of IEEE*, vol. 83, no. 4, pp. 687–693, April 1995.

[2] S. Gitzenis and N. Bambos, "Joint task migration and power management in wireless computing," *IEEE Transactions on Mobile Computing*, vol. 8, no. 9, pp. 1189–1204, 2009.

[3] S. Jayashree and C. Ram Murthy, "A taxonomy of energy management protocols for ad hoc wireless networks," *Communications Magazine, IEEE*, vol. 45, no. 4, pp. 104–110, April 2007.

[4] Z. Li, C. Wang, and R. Xu, "Computation offloading to save energy on handheld devices: a partition scheme," in *CASES '01: Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems.* New York, NY, USA: ACM, 2001, pp. 238–246.

[5] S. Gurun, R. Wolski, C. Krintz, and D. Nurmi, "On the efficacy of computation offloading decision-making strategies," *International Journal of High Performance Computing Applications*, vol. 22, no. 4, pp. 460–479, 2008.

[6] C. Xian, Y.-H. Lu, and Z. Li, "Adaptive computation offloading for energy conservation on battery-powered systems," *Parallel and Distributed Systems, International Conference on*, vol. 1, pp. 1–8, 2007.

[7] X. Gu, A. Messer, I. Greenberg, D. Milojicic, and K. Nahrstedt, "Adaptive offloading for pervasive computing," *IEEE Pervasive Computing*, vol. 3, no. 3, pp. 66–73, 2004.