Optimal Joint Scheduling and Cloud Offloading for Mobile Applications

S. Eman Mahmoodi, *Student Member, IEEE,* R. N. Uma, *Member, IEEE, Member, ACM,* and K. P. Subbalakshmi, *Senior Member, IEEE*

Abstract—Cloud offloading is an indispensable solution to supporting computationally demanding applications on resource constrained mobile devices. In this paper, we introduce the concept of wireless aware *joint* scheduling and computation offloading (JSCO) for multi-component applications, where an optimal decision is made on which components need to be offloaded as well as the scheduling order of these components. The JSCO approach allows for more degrees of freedom in the solution by moving away from a compiler pre-determined scheduling order for the components towards a more wireless aware scheduling order. For some component dependency graph structures, the proposed algorithm can shorten execution times by parallel processing appropriate components in the mobile and cloud. We define a net utility that trades-off the energy saved by the mobile, subject to constraints on the communication delay, overall application execution time, and component precedence ordering. The linear optimization problem is solved using real data measurements obtained from running multi-component applications on an HTC smartphone and the Amazon EC2, using WiFi for cloud offloading. The performance is further analyzed using various component dependency graph topologies and sizes. Results show that the energy saved increases with longer application runtime deadline, higher wireless rates, and smaller offload data sizes.

Index Terms—Joint scheduling–offloading, mobile cloud computing, computation offloading, scheduling.

1 Introduction

Cloud offloading has become a recognized solution for delivering computationally intensive applications (e.g., video-intensive games, computer vision-based applications [1], and real-time visual information reporting) on resource-constrained mobile devices [2], [3]. Typically, energy and time (or delay) constraints have played a strong role in determining offloading policies. Recently, we argued that the burden placed on the wireless networks supporting this offloading must also be taken into consideration [4] when developing offloading strategies. In [4], we proposed an optimal offloading policy for applications with sequential component dependency graphs and multi-radio enabled mobile devices, that minimizes the energy consumed by the mobile device such that overall execution time of the application will be below a given threshold while simultaneously determining optimal percentage of data (associated with computation offloading) to be transferred via each of the multiple wireless interfaces. In this paper, we address the problem of cloud offloading for mobile applications with *arbitrary* dependency graphs rather than sequential dependencies or pre-determined compiler generated schedule order. To this end, we must consider wireless-aware scheduling of the application components jointly with the offloading strategy. We optimally maximize a net utility function, which trades-off the energy saved at the resource constrained device with the time and energy costs involved in offloading while meeting the precedence constraints and execution deadline of the application in single radio enabled mobile devices. *To the best of our knowledge, this is the first work that proposes joint scheduling–offloading for mobile applications. By optimizing the scheduling of the individual components along with cloud offloading decisions, taking into account the wireless parameters, allows for an overall better solution compared to optimizing only the offloading decisions using a pre-determined compiler-generated schedule order of execution for the individual components. Besides, using the general dependency graphs (without imposing a sequential ordering for processing) and an optimal joint scheduling–offloading scheme can potentially allow for parallel scheduling of components in the mobile and cloud at the same time, thus reducing time to completion for the application.*

Cloud offloading can be interpreted as data flow offloading in networking applications [5] or offloading computationally intense tasks to the cloud [6] or cloudlet [7], which is a self-managing data center in the layer of network infrastructure [8]. In this paper, we refer to computation offloading to the cloud. Existing work on computation offloading to cloud resources can be classified into three types: (*i*) ones that offload all of the application to a cloud [9], [10]; (*ii*) those in which "all or nothing offloading" is applied where either the entire application is offloaded to the cloud or executed locally, typically depending upon which is more energy efficient for the mobile device [11]; and (*iii*) partial offloading strategies where some of the component tasks are offloaded while the others are executed locally [4], [12],

- *S.E. Mahmoodi and K.P. Subbalakshmi are with the Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, NJ,* 07030.
  E-mails: {smahmood, ksubbala}@stevens.edu
- *R.N. Uma is with the Department of Mathematics and Physics, North Carolina Central University, Durham, NC,* 27707.
  Email: ruma@nccu.edu

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560808, IEEE Transactions on Cloud Computing

[13], [14], [15]. In partial offloading, the application can be coarsely partitioned into components [4], [12], [16], [17] or a more fine-grained offloading can be achieved [18] by using method-level partitioning as in MAUI [13], ThinkAir [14] and CloneCloud [19].

Compared to the related work in Section 3, this paper has several contributions. This is the first work, to the best of our knowledge, that combines offloading and scheduling decisions in the presence of arbitrary component dependencies (precedence constraints of components). To enable capturing this multi-dimensional decision aspect of this problem, a mathematical formulation is best suited. Hence a key technical contribution of this paper is the mathematical formulation of the component offloading problem that arises in real mobile applications as an optimization problem using integer linear programming formulations. The resulting mathematical formulation is non-trivial. Specifically first, we model the required joint offloading–scheduling decision variables, latencies and the energy saved by cloud offloading. Then, we provide a mathematical analysis for the optimization problem for joint wireless-aware scheduling of the mobile application and cloud offloading. Note that the optimization problem is linearized in order to take benefit of linear programming (LP) for obtaining the optimal solution. Third, real data are measured from an HTC smartphone using real and random generated mobile applications, WiFi radio interface for computation offloading, and Amazon Elastic Compute Cloud (EC2) for remote execution. We identify the optimal solution under these real data measurements using IBM CPLEX optimizer [20]. Finally, we derive a comprehensive performance analysis of this work compared with upper and lower bounds for dependencies of applications, the number of application components, topology of application component dependency graphs (CDGs), application runtime, and wireless parameters such as rates, latencies, and data sizes.

The rest of this paper is organized as follows. The CDGs of mobile applications and related work are respectively expressed as two important backgrounds of the paper in Sections 2 and 3. Then, we model the Joint Scheduling and Computation Offloading (JSCO) scheme and formulate the optimization problem regarding to the constraints for scheduling, delay, runtime and completion deadlines in Section 4. In Section 5, we present experiments and simulations to evaluate the performance of the proposed optimal strategy. Finally, Section 6 presents the conclusion and future work of the paper.

2 Background: Component Dependency Graphs

All of the prior work discussed above on partial cloud offloading consider mobile applications with sequential component dependencies or component scheduling order that is predetermined by a compiler. In general, components in a real life application can have arbitrary dependency graphs and potentially, an overall better solution can be obtained by designing a *joint scheduling–offloading* policy for the components where the scheduling order of the components is also cognizant of the wireless network supporting the offloading.

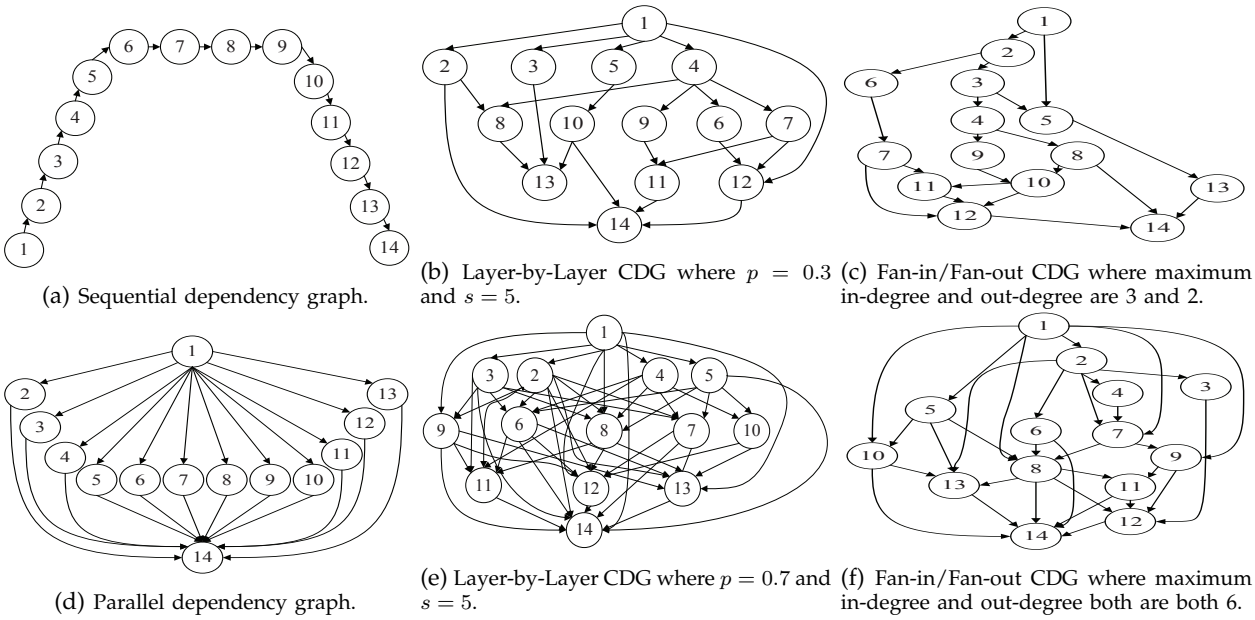
Component dependency graphs (CDGs) of mobile applications must satisfy these general properties: (*i*) each component should have at least in-degree of one (except the first component, which has in-degree of zero); (*ii*) components should have at least out-degree of one (except the last component N, which has out-degree of zero); (*iii*) all of the components should have at least one direct or indirect path from component 1 so that they are dependent on the common starting point of the application (typically executed on the mobile); (*iv*) all of the components should have at least one direct or indirect path to component N (the last component) and (*v*) in the adjacency matrix (M) of the CDG, all the diagonal elements are zero, because there is no self-dependency.

Fig. 1 presents different types of CDGs for an N-component application (N=14): (*i*) sequential dependency graphs where all the components are sequentially dependent (Fig. 1a); (*ii*) parallel dependency graph where only component 1 must be executed before components 2 to $N-1$. In addition, these components are only required to transfer their output data to component N (Fig. 1d); (*iii*) random Layer-by-Layer graph (Figures 1b, 1e); and (*iv*) random Fan-in/Fan-out graph (Figures 1c, 1f) [21]. In Layer-by-Layer CDGs, a random number of nodes is generated for each of the layers and edges are added with a probability p going from a node in an earlier layer to a node in one of the successive layers. In Fan-in/Fan-out CDGs, the Fan-in/Fan-out ratio of each node is constrained to the given threshold. Since usually mobile-initiated applications must start on the mobile device and have an output display on the mobile device, the first and last components are processed in the mobile device. Note that the parallel and sequential dependency graphs show the lowest and highest dependencies between components respectively and can be used to obtain the lower and upper ranges for the cost of offloading on applications exhibiting these extremes of CDGs.

3 Related Work

Time scheduling of the application components is studied in *eTime* [9] and [22] in which a pre-determined compiler-generated order of execution for the application components is considered and all the component tasks are offloaded for remote execution. *eTime* explores an energy-delay trade-off in scheduling the required data transmissions for offloading (entire computations of application) such that the queue stability of the wireless interface is satisfied and offloading is done when the wireless connectivity is sufficiently good. A

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560808, IEEE Transactions on Cloud Computing



(a) Sequential dependency graph.

(b) Layer-by-Layer CDG where $p = 0.3$ and $s = 5$.

(c) Fan-in/Fan-out CDG where maximum in-degree and out-degree are 3 and 2.

(d) Parallel dependency graph.

(e) Layer-by-Layer CDG where $p = 0.7$ and $s = 5$.

(f) Fan-in/Fan-out CDG where maximum in-degree and out-degree both are both 6.

Fig. 1: Examples of various CDGs for the mobile applications ($N = 14$).

scheduling policy for partially offloading the sequence of fine-grained tasks with serial CDG (as in Fig. 1a) is proposed in [15] such that the application execution time is guaranteed. While these works that use fine-grained method partitioning for partial offloading are limited in input/environmental conditions in the offline pre-processing and need to be bootstrapped for every new application, our work, which uses component scheduling, does not involve this problem. Existing component-based mobile cloud offloading strategies, such as DOA [17] and MACS [23], are not designed for parallel processing simultaneously via the mobile device and cloud because they use a pre-determined order of traversal of the application CDG. Our proposed scheme has this flexibility. Another scheduling scheme to minimize the total energy consumption in a multi-user network is studied in [24] where a centralized broker partially offloads sequential tasks to the cloud. Thus, a centralized strategy is required to perform a two-hop offloading where the broker is an intermediary between the mobile user and the cloud. However, using scheduling strategies based on arbitrary CDGs extends the number of applications to be used for partial cloud offloading. In [16], sequential scheduling of the computational tasks is considered in both single-channel and multi-channel communications. The objective is to minimize the energy consumed while simultaneously meeting the delay constraints of the application. However, wireless-aware scheduling of the application components provides higher energy and spectrum efficiencies in cloud offloading strategies.

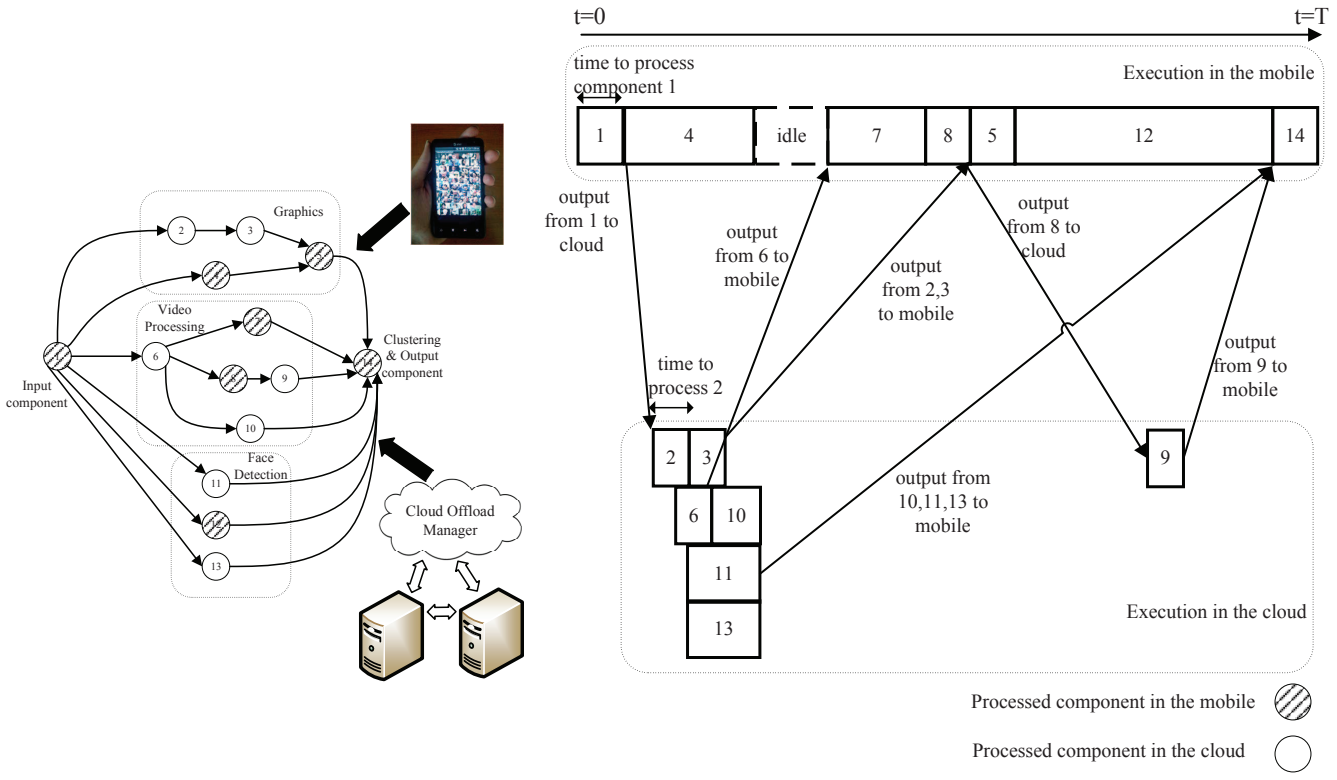
4 Proposed Scheduling Model for Mobile Cloud Offloading

In the mobile cloud offloading model considered in this work, the mobile device has access to a cloud server for computation offloading, and the cloud server is endowed with parallel processing capabilities. We additionally make the following assumptions: (*i*) the multi-component mobile application that is utilized by the mobile user is also installed on the cloud server; and (*ii*) mobile broadband connectivity does not change during the application processing time while the wireless interface may provide different rate and delay values. Note that in the second assumption, we consider that application processing time is not large, and most of the related works have also assumed this condition [13], [14], [16], [17], [24], [23]. Following these assumptions, we show a mobile cloud offloading model example of a 14-component application in Fig. 2b. Note that this 14-component topology is the same as one of the applications we used in the performance analysis section.

4.1 Multi-Component Application Example for the Scheduling–Offloading Model

Here we used a video navigation application, which involves graphics [25], face detection [26], camera preview, and video processing [27], running on an HTC Vivid smartphone. Fig. 2a shows the dependency graph for 14 components of the application. The link connection between components i and j shows that the output data from component i is required as input by component j, and d_{ij} represents the required data size for transferring from i to j. We observe that this dependency could be either sequential (like the dependencies between components 1-2-3-5-14) or parallel (like the component dependencies between 1-11-14, 1-12-14, and 1-13-14). In Fig. 2b, an example of joint scheduling–offloading of the components based on time, place of processing, and dependency among the components is illustrated. If a component is scheduled for offloading to the cloud, the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560808, IEEE Transactions on Cloud Computing



(a) CDG of the application. (b) Scheduling–Offloading model.

Fig. 2: Scheduling model for cloud offloading in a 14-component mobile application with a general CDG.

energy consumption for processing will be saved by remote execution. In addition, the time for processing the component decreases significantly by remote execution (compare the times taken to process components by the mobile device and the cloud in Fig. 2b). Moreover, some components can be processed in parallel by the cloud (components 2, 6 and components 3, 10). However, the cost of cloud offloading should also be considered in the scheduling–offloading decisions: (*i*) the costs of delay and energy consumed by offloading as a function of data size for transferring (e.g., component 11 has very large data for transferring so it takes a longer time for communication); and (*ii*) the cost of the idle state as the mobile waits to receive the required output data from the cloud (between components 4 and 7). Thus, a *smart* scheduling strategy for mobile offloading based on *energy-time* trade-off is required.

4.2 Proposed Optimal Joint Scheduling & Computation Offloading Scheme (JSCO)

In this section, we present the formulation of our problem as an integer linear program. For each time period $(t-1, t]$ denoted by time slot index t, we define decision variable, x_{ljt}, which indicates whether component j completes processing at time slot t on the mobile ($l = 0$) or on the cloud ($l = 1$). This decision variable captures the multi-objective requirement of mobile communication applications to provide "*anywhere, anything, anytime*" service.

The processing indicators in the mobile and cloud are respectively given by $m_j = \sum_{t=1}^{T} x_{0jt}$, $c_j = \sum_{t=1}^{T} x_{1jt}$, $\forall j$, where T is the number of time periods to complete processing the application. Also τ_{ij}^{cm} denotes the time (the number of time slots) to transfer data from component i to j when $i \prec j$, and j is processed on the mobile and i is processed on the cloud. τ_{ij}^{cm} includes the product $m_j c_i$ where i is processed on the cloud and j is processed on the mobile. In order to make the optimization problem linear, this quadratic term of two binary decision variables is replaced by a new variable z_{ji} where z_{ji} is the component transferring indicator. z_{ji} gets 1 if the output data of component j (component j is executed in the mobile) is offloaded from the mobile device to the cloud where component i ($i \prec j$) will be executed. Otherwise, it gets 0. This parameter must satisfy the following four constraints [28]: $z_{ji} \leq m_j$, $z_{ji} \geq 0$, $z_{ji} \leq c_i$, $z_{ji} \geq c_i - (1 - m_j)$, $\forall i, j$. Thus, the quadratic term of two decision variables is converted to a new decision variable so that the optimization problem still remains linear. Similarly, τ_{ij}^{mc} denotes the time (the number of time slots) to transfer data from i to j when i is processed on the mobile device and j is processed on the cloud and includes $m_i c_j$ which is denoted by the variable z_{ij}. Now the times for transferring from mobile to cloud and cloud to mobile are respectively given as $\tau_{ij}^{\mathrm{cm}} = \alpha_{ij} z_{ji} \frac{d_{ij}}{R_{\mathrm{d}}}$, $\tau_{ij}^{\mathrm{mc}} = \alpha_{ij} z_{ij} \frac{d_{ij}}{R_{\mathrm{u}}}$, $\forall i, j$, where α_{ij} is the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560808, IEEE Transactions on Cloud Computing

TABLE 1: Parameter Definitions.

Parameters	Definitions
N	number of components in the application.
T	number of time periods to complete processing the application.
t	time index for period (t-1,t].
m_j	mobile execution indicator for component j.
c_j	cloud execution indicator for component j.
x_{ljt}	a binary indicator which equals to 1 if component j completes processing at time t on processing system l and otherwise equals to 0.
α_{ij}	dependency indicator: 1 if component i must be processed before j and 0 otherwise.
z_{ij}	component transferring indicator, which equals to $m_i c_j$.
d_{ij}	size of data required by component j from component i.
q_j^{m} (q_j^{c})	time to process component j in the mobile (cloud).
τ_{ij}^{mc}	time required to transmit data from component i executing in the mobile to component j executing in the cloud.
τ_{ij}^{cm}	time required to receive data from component i executing in the cloud to component j executing in the mobile.
ν_k	time to process component k either on mobile or cloud.
E_{com}	the total energy consumed by the mobile device for communication.
P_{ac}	active power of the mobile while processing a component.
P_{Tx} (P_{Rx})	power consumption of the mobile to transmit (receive) required data.
R_{u} (R_{d})	average uplink (downlink) rate of the wireless radio interface.

dependency indicator, and gets 1 if component i must be processed before j and 0 otherwise. d_{ij} is the size of data required by component j from component i, and R_{u} (R_{d}) is the average uplink (downlink) rate of the wireless radio interface. Note that τ_{ij}^{cm}, τ_{ij}^{mc} will be zero if $i = j$, or if i does not precede j, or if i and j are both processed on the cloud, or both processed on the mobile device. In addition, the energy consumed for communication due to cloud offloading the components is modeled by

$$E_{\mathrm{com}} = P_{\mathrm{Tx}} \sum_{i=1}^{N} \sum_{j=1}^{N} \tau_{ij}^{\mathrm{mc}} + P_{\mathrm{Rx}} \sum_{i=1}^{N} \sum_{j=1}^{N} \tau_{ij}^{\mathrm{cm}}. \quad (1)$$

The objective function in the optimization problem over decision variables (x_{ljt}, z_{ij}, $l \in \{0,1\}$, $i,j = 1,\ldots,N$, $t = 1,\ldots,T$) for the mobile cloud offloading scheme is mathematically formulated as

$$\max\{\sum_{j=1}^{N} P_{\mathrm{ac}} c_j q_j^{\mathrm{m}} - E_{\mathrm{com}}\}. \quad (2)$$

Eqn. (2) shows the maximization of the energy saved through remote execution. This energy saved is essentially the energy cost if the offloaded components had been executed locally minus the cost of communication energy.

Besides the constraints for quadratic parameter, the following constraints should be satisfied in the optimization problem with the objective function given by Eqn. (2):

Runtime deadline constraint: The multi-component application has a time deadline, which should be satisfied. This constraint is given by $0 < \sum_{t=1}^{T} t x_{0Nt} \leq T$, $\forall t$, where $\sum_{t=1}^{T} t.x_{0Nt}$ denotes the completion time slot for processing the last component (N) on the mobile ($l = 0$). This time slot should be equal or less than T.

Each component be processed only once: Each component is processed either in the mobile or cloud, which can be written as

$$m_j + c_j = 1 \qquad \forall j. \quad (3)$$

Precedence constraint: This constraint shows that component k is required to begin processing no earlier than the completion time of component j where $j \prec k$. The constraint is expressed as

$$\sum_{l=0}^{1} \sum_{s=1}^{t+\nu_k+\tau_{jk}^{\mathrm{cm}}+\tau_{jk}^{\mathrm{mc}}} x_{lks} \leq \sum_{l=0}^{1} \sum_{s=1}^{t} x_{ljs}, \quad \text{if } j \prec k, t = \nu_j, \ldots, T - \nu_k - \tau_{jk}^{\mathrm{cm}} - \tau_{jk}^{\mathrm{mc}}, \quad (4)$$

where ν_k is the time to process component k either on the mobile or cloud, and is given by $\nu_k = m_k q_k^{\mathrm{m}} + c_k q_k^{\mathrm{c}}$. Based on Eqn. (3), ν_k will include either the cloud processing time slots for component k or the mobile processing time slots for component k, but not both. Here in constraint (4), in order for k to be completed after the time t plus the time for possible data transferring from j to k ($\tau_{jk}^{\mathrm{cm}} + \tau_{jk}^{\mathrm{mc}}$), plus the time for processing component k (ν_k), component j must be completed by time t, $\forall t$.

Serial computation at the mobile device: The processed components in the mobile are required to be executed in serial. Thus, for each time interval $[t-1, t)$ we can have at most one component for processing in the mobile, which can be written as $\sum_{j=1}^{N} \sum_{s=t}^{\min\{t+\nu_j-1,T\}} x_{0js} \leq 1$, $\forall t$.

Completion deadline: Each component k must be completed only after the completion of each of its precedent components like j, plus the time (slots) to process component k itself, and the time slots to transfer required data to the execution site of k if j is not on that same site. This constraint is given by $\sum_{l=0}^{1} \sum_{t=1}^{T} t x_{ljt} + \tau_{jk}^{\mathrm{cm}} + \tau_{jk}^{\mathrm{mc}} + \nu_k \leq \sum_{l=0}^{1} \sum_{t=1}^{T} t x_{lkt}$, if $j \prec k$, $k = 1,\ldots,N$. Also, decision variables should be $0-1$, $x_{ljt} \in \{0,1\}$,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560808, IEEE Transactions on Cloud Computing

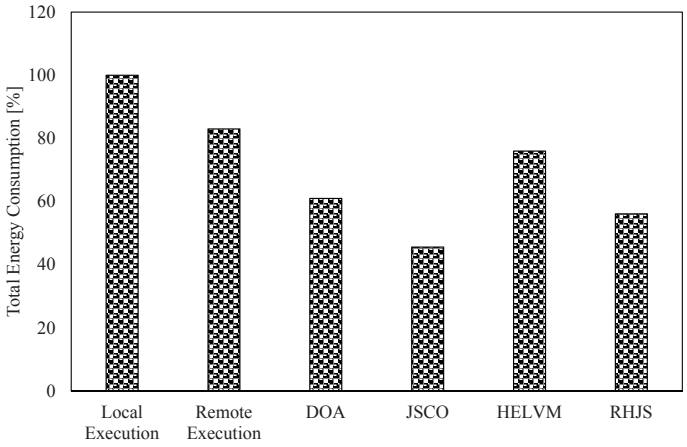


Fig. 3: Total energy consumed by the mobile device for the proposed and classical schemes, normalized to the energy consumed by local execution (using the face recognition application in [30]).

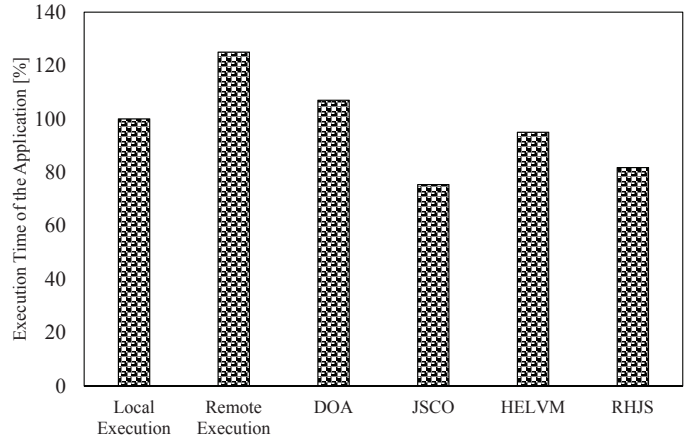


Fig. 4: Total execution time of the application for the proposed and classical schemes, normalized to the execution time by local execution (using the face recognition application in [30]).

$l \in \{0,1\}, \forall j,t$, and get zero values while the coordinated component has not been processed yet, which is written as $x_{ljt} = 0$, $l \in \{0,1\}$, $\forall j, t = 1, \ldots, \nu_j - 1$.

4.3 Scheduling Overhead

Since we have a linear optimization problem, the number of constraints plays the important role in scheduling overhead because number of constraints affects memory usage more than the number of variables [29]. In this work, the number of constraints is $(6+T)N^2+4N+T+2$, which is a function of application runtime and number of components (order of complexity is O(TN^2)). Also, the number of variables is N^2+2TN (order of complexity is O(TN). However, we do not experience schedule overhead in the offloading scenario because (*i*) the strategy will be executed in the cloud server where the RAM is high enough, and (*ii*) JSCO is not required for a real-time scenario while we assume fix wireless parameters.

5 Performance Analysis

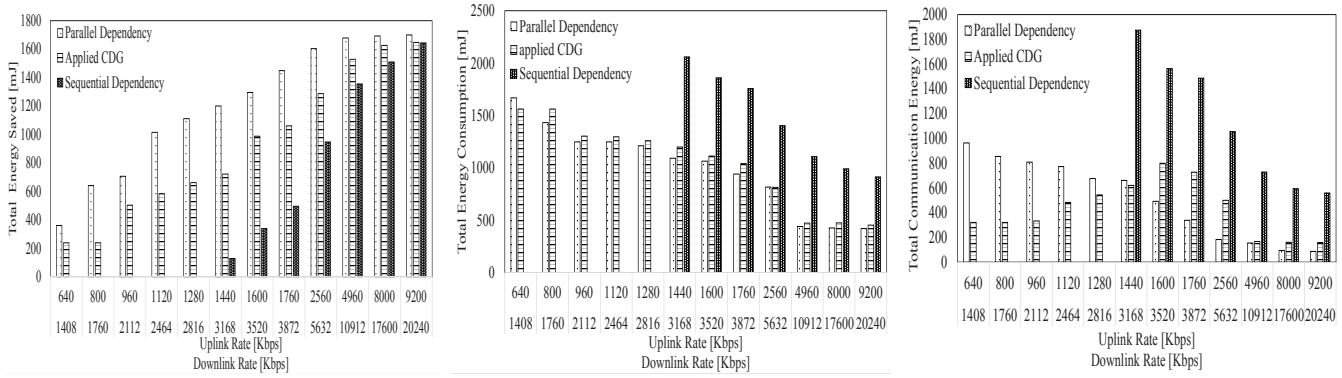
In this section, we first discuss the performance of the proposed JSCO scheme in comparison with the related works using an application in [30], and also based on a real application for which we made real data measurements. This is the 14-component application whose CDG was presented in Fig. 2a. To further the understanding of our model's adaptability and scalability, we considered some randomly generated CDGs whose layered structure and Fan-in/Fan-out ratio could be controlled.

5.1 Real Data Measurements and Simulation Setup

An HTC Vivid smartphone with a 1.2GHz dual-core processor and WiFi radio interface were used to gather real data. To test the performance of the proposed optimal scheme, a multi-component video navigation application was used where video processing, face detection, graphics, and clustering were the main features. In all, 14 components were used, four of which are related to the graphics feature, three are for the face detection feature, six are for video processing, and one is for clustering. Note that the first and last components are executed locally so that the input-output of the application is accessed by the mobile user. In addition, graphics library tools from the OpenGL mobile Android applications were used [25]; face detection was taken from [26]; and all the video processing features were obtained from [27]. The CDG of this application is illustrated in Fig. 2a. The execution times of the components in the HTC phone and the cloud, uplink and downlink rates, delay at the WiFi interface were measured. The Amazon Elastic Compute Cloud (Amazon EC2) was used as the cloud computing server. The average transmission and reception power levels of the mobile device for WiFi service were 257.83 and 123.74mW, respectively. The active and idle power levels of the phone were 644.9 and 22mW, respectively. The power consumption of the last component in the mobile device was 55mW. These power measurements were obtained using the "CurrentWidget: Battery monitor" application [31]. The average wireless service rates for WiFi, obtained using the TCPdump tool, were 0.80Mbps for the uplink transmission and 1.76Mbps for the downlink transmission, respectively. The local execution time for the 14 components were measured as [30 340 345 125 30 80 70 30 185 125 650 571 904 56] ms. Because processing of the components in the mobile device is performed in serial, application runtime in the local execution equals the sum of the processing times for the 14 components (3541ms). Also note that here each time period $(t-1, t]$, $\forall t$, is set to 1ms.

The obtained real data measurements were used in the linear programming proposed in Section 4.2 with the objective function as shown in Eqn. (2) subject to the expressed constraints. We used the IBM CPLEX optimizer [20] to solve the integer linear problem, which is known to be NP hard. Also, the JSCO strategy is scheduled at the cloud server.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560808, IEEE Transactions on Cloud Computing



(a) Total energy saved. (b) Total energy consumption. (c) Total communication energy.

Fig. 5: Total energy for the 14-Component application versus uplink and downlink rates in WiFi while T=3s, P_{Tx}=257.83mW, P_{Rx}=123.74mW.

5.2 Comparison of JSCO with State of Art

We compare our proposed optimal work (JSCO) to (1) no offload (local) execution where all the components are executed locally; (2) all offload (remote) execution where all the components are offloaded to the cloud; (3) the dynamic offloading algorithm (DOA) in [17], which uses an energy efficient partial offloading strategy; (4) HELVM algorithm from [32], which provides runtime offloading services; and (5) a heuristic algorithm that is the revised HEFT [33] for joint scheduling (RHJS) tasks on multiple cores used in [34]. In the simulations for this subsection, a face recognition application with 10 sequential components was utilized [30]. The wireless network parameters in [35] are used such that exactly the same parameters used for the simulation of DOA in [17] were used for all the other schemes.

In Fig. 3, we compare the total energy consumption of the proposed scheme (JSCO) with the 5 schemes. This comparison is normalized to the scheme with local execution of all the components. It is observed that JSCO consumes 54%, 37%, 16%, 30%, and 11% less energy in comparison to the schemes using local execution, remote execution, DOA, HELVM, and RHJS, respectively.

Fig. 4 shows the time to run the application [30] for the 6 schemes. This comparison is also normalized to the scheme with local execution of all the components. We see that by using the optimal JSCO scheme, the application will be executed 25%, 49%, 32%, 19%, and 5% faster in comparison to the schemes using local execution, remote execution, DOA, HELVM, and RHJS, respectively. Thus, JSCO is a joint energy and time efficient scheme in comparison to the other 5 schemes.

5.3 Simulations for the Real Mobile Application

In this subsection, we analyze the performance of the proposed JSCO scheme using the real 14-component application (referred to as "applied CDG") w.r.t the critical parameters of rate, time, and data size. We compare and contrast the performance of our strategy on the real 14-component application with arbitrary dependencies (Fig. 2) against a 14-component application with fully parallel dependencies (Fig. 1d) and a 14-component application with fully sequential dependencies (Fig. 1a). Since the parallel and sequential dependency graphs show, respectively, the lowest and highest dependencies between components, lower and upper bounds for the cost of offloading could be obtained for the applied CDG.

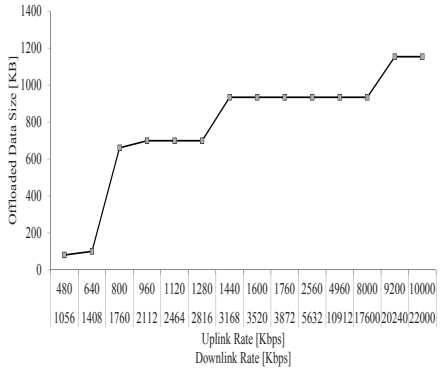


Fig. 6: Offloaded data size of the application components versus rates of the WiFi link while T=3s, P_{Tx}=257.83mW, P_{Rx}=123.74mW.

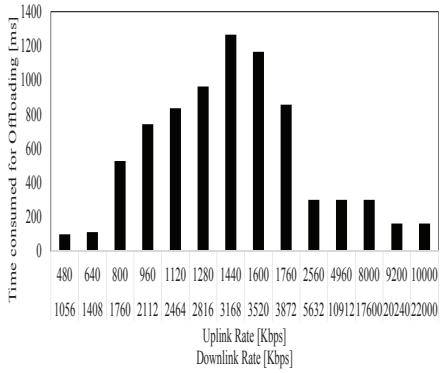


Fig. 7: Time consumed for offloading versus rates of the WiFi link while T=3s, P_{Tx}=257.83mW, P_{Rx}=123.74mW.

Rate Plots: Fig. 5 shows the total energy values for several uplink and downlink rates of the WiFi interface provided for cloud offloading. Fig. 5a presents the total energy saved through remote execution (the objective

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560808, IEEE Transactions on Cloud Computing

function in Eqn. (2)) versus wireless rates. We see that while rates increase, more energy is saved by the mobile device with cloud offloading. This is expected, because with higher rates, data communication is no longer a bottleneck and it is more energy efficient to offload as many components as possible to the cloud. More energy is saved in the parallel dependency graph, while less energy is saved in the sequential dependency graph. We observe that in the sequential dependency graph, no energy can be saved by cloud offloading for lower ranges of rates, and the application cannot be processed with these low rates in three seconds (the time for local execution is 3541ms). However, in the higher ranges of rates (uplink (downlink) rate= 9200 (20240Kbps)), most of the components are offloaded to the cloud for computations in all three CDGs. Thus, the performances of these three are closer to each other when the wireless rates increase. In Fig. 5b, the total energy consumed by the mobile device (summation of active energy while the mobile device is executing components locally, communication energy, and idle energy while the mobile's processor is not executing any component) is plotted. We see that less total energy is consumed by the mobile device when WiFi rates increase. Moreover, Fig. 5c illustrates the energy consumed by communication, E_{com} (given in Eqn. (1)), versus wireless rates. It is observed that the energy consumed by communication decreases with an increase in rates for the sequential and parallel dependency graphs. Although this is true for the applied CDG in higher rate ranges, more energy is consumed by offloading while rates increase in the lower ranges. The reason is that more computations are offloaded when rates increase so more energy is required for offloading, while in the higher ranges of rates, the time to offload decreases thereby decreasing the communication energy. Note that the application with sequential dependency cannot be executed until rates reach $1440/3168$Kbps. In the lower rate ranges, wireless delay is high, and offloading is not preferred. On the other hand, local execution takes 3541ms when the application deadline, T, is set to 3000ms in the simulations for this figure. Therefore, the scheme using sequential dependency graph is not plotted at lower rates because the application cannot be executed in $T = 3000$ms.

Figures 6 and 7 depict the offloaded data size and the time span for communication versus uplink and downlink rates for the applied CDG. It is observed in Fig. 6 that while rates increase, more data is transferred for cloud offloading. More components for offloading leads to the consumption of more energy and time for offloading, as shown in Figures 5c and 7, respectively. For rates higher than 1600Kbps uplink and 3520Kbps downlink in Fig. 6, we observe that the data size for offloading does not change much; however, the time for offloading decreases. This results in a corresponding decrease in the energy consumed for communication.

Time Plots: Figures 8a, 8b, and 8c respectively plot the total energy saved, total energy consumption, and the energy consumed by communication versus execution time of the application for the three different CDGs considered–sequential, applied and parallel. When more time is allotted for the execution of the application, cloud offloading is preferred and leads to a decrease in energy expenditure by the mobile device.

Figures 5 and 8 show that using the JSCO scheme (the scenario where the applied CDG is used) works better than using an optimal offloading scheme that uses a compiler pre-determined sequential traversal of an arbitrary CDG (the scenario where the sequential dependency is used). Examples of sequential traversals of arbitrary CDGs include [13], [36]. Specifically, we see from Fig. 8 that the processing of an application with sequential traversal CDG can be completed in no less than 3300ms, while the application with applied CDG can be processed in 2400ms and the application with parallel CDG can be processed in 2000ms (rates are set to $800/1760$Kbps). In addition, the application with sequential dependency cannot be executed until rates reach $1440/3168$Kbps, whereas the applied CDG is processed at much lower rates, $640/1408$Kbps, while T is set to 3000ms (Fig. 5).

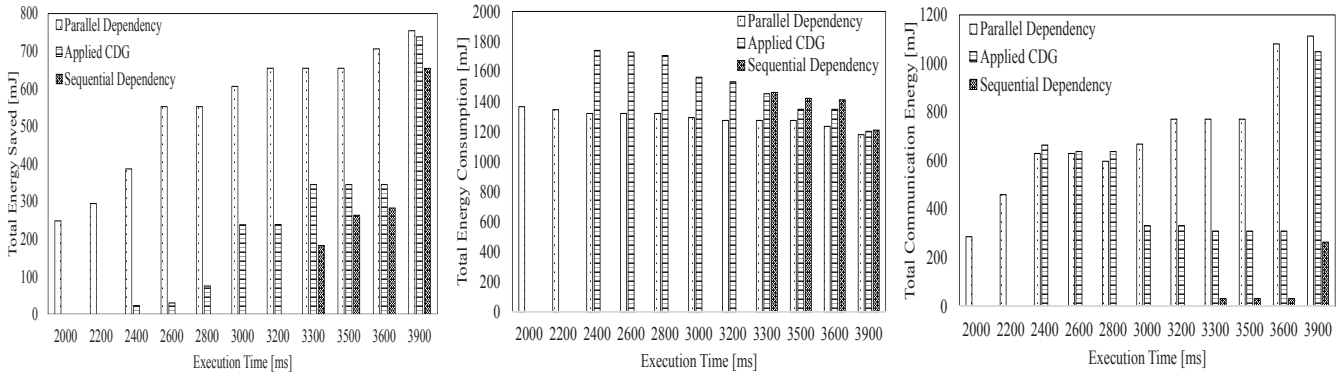
We consider another metric, the number of transitions, where a transition is a data transfer between the mobile device and the cloud. In Fig. 9, the number of transitions between the mobile device and the cloud is plotted against the execution time for the applied CDG. We see that for the simulations where $T \geq 2900$ms, the number of transitions between the mobile and cloud decreases from six to four. Moreover, Fig. 10 illustrates that the size of offloaded data decreases while the application execution time increases. These two figures show that computation offloading decreases while the execution time increases. Therefore, the communication energy decreases while the execution time increases, as shown in Fig. 8c for the applied CDG.

The Data Plot: We next look at the impact on energy consumption and savings when the amount of data to be transferred increases. Here the required data transfer for face detection components is increased from 21.4KB to 2.2MB to consider the performance of total energy as a function of the data size required for transition. In Fig. 11, we see that, as expected, while the data size for transferring increases, more energy is consumed for communication, less energy is saved, and more energy is consumed by the mobile device.

5.4 Simulations for Variety of Component Dependencies

So far, the system performance was analyzed based on the fixed CDG from the 14-component video navigation application shown in Fig. 2, as well as the two extreme cases of fully sequential CDG and fully parallel CDG. In this section, we consider the performance of the proposed system based on two different categories of random CDGs: (*i*) Layer-by-Layer, and (*ii*) Fan-in/Fan-out, as explained in Section 2. Since we use random

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560808, IEEE Transactions on Cloud Computing



(a) Total energy saved. (b) Total energy consumption. (c) Total communication energy.

Fig. 8: Total energy versus execution time (T) while R_{u}=0.8Mbps and R_{d}=1.76Mbps.

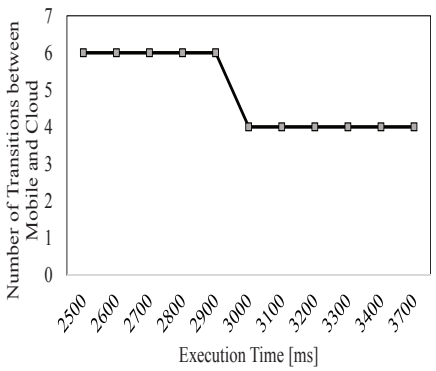


Fig. 9: Number of transitions between the mobile and cloud for offloading in correspondence with execution time (T) while R_{u}=0.8Mbps and R_{d}=1.76Mbps.

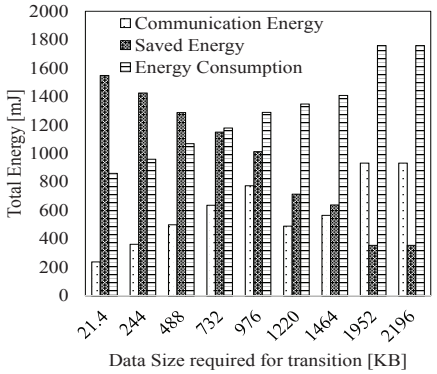


Fig. 11: Total energy versus required data size for transition of components while T=3s, R_{u}=0.8Mbps and R_{d}=1.76Mbps using the applied CDG.

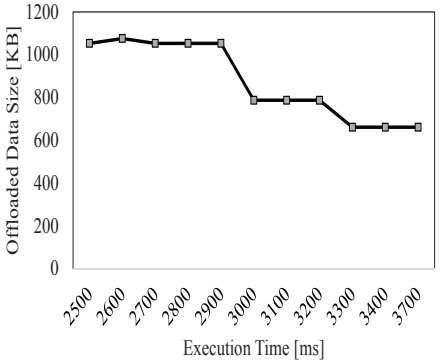


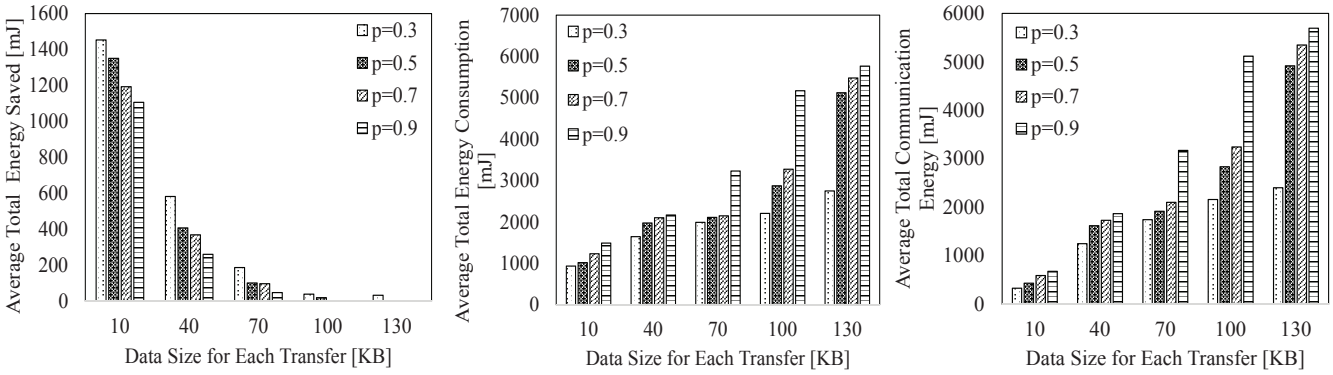
Fig. 10: Allocated data size for cloud offloading in correspondence with execution time (T) while R_{u}=0.8Mbps and R_{d}=1.76Mbps.

CDGs in this subsection, the simulations for each data point are run over three CDGs and the average of these three values is plotted. Each CDG that we generate is constrained to have only 14 components for comparison purposes.

Figures 12 and 13 show the performance of the proposed JSCO scheme for randomly generated Layer-by-Layer CDGs. In Fig. 12, the average total energy saved through remote execution, the average total energy consumed by the mobile and the average total energy for communication are plotted against the size of data transferred. These bar graphs are compared as a function of the probability of edge connections (p). When this probability increases, more components are dependent on each other, and the density of the CDG increases. Therefore, the energy consumed by cloud offloading increases (Fig. 12c), and the energy saved through remote execution decreases (Fig. 12a). Moreover, it can be observed that when data size for transferring the components increases, the total energy consumed by the mobile device and the energy consumed for communication increase (Figures 12b, 12c), and the energy saved through remote execution decreases (Fig. 12a). Also note that for high values of p and size of data transfer, the energy costs of offloading are so high that the energy saved through remote execution gets closer to zero (as shown in Fig. 12a).

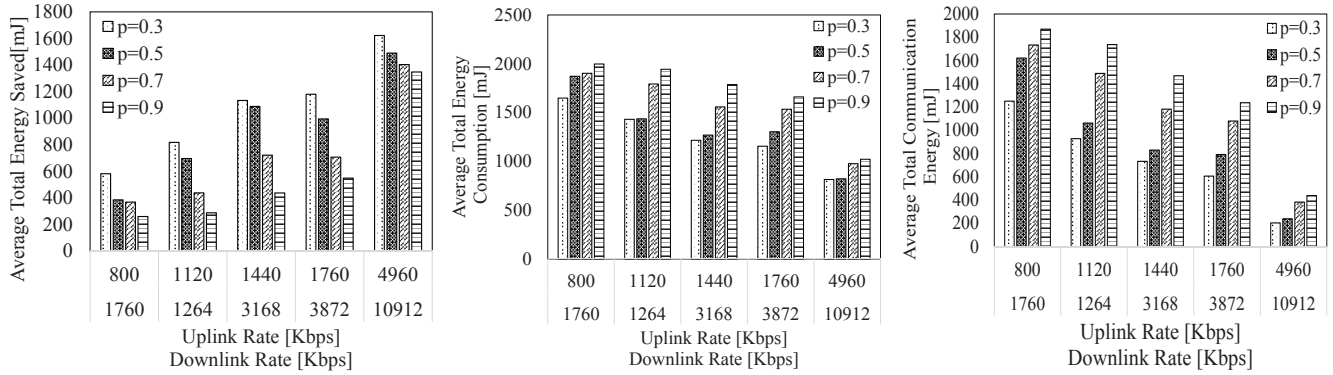
In Fig. 13, the average total energy saved through remote execution, the average total energy consumed by the mobile, and the average total energy for communication are plotted against uplink and downlink rates. These bar graphs are also compared as a function of the probability of edge connections. We can observe that while the wireless rates increase, the energy consumed by offloading decreases (Fig. 13c), the energy saved

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560808, IEEE Transactions on Cloud Computing



(a) Average total energy saved. (b) Average total energy consumption. (c) Average total communication energy.

Fig. 12: Average total energy versus required data size for transferring each component in the apps with Layer-by-Layer CDG ($s = 5$) and 14 components while T=3s, P_{Tx}=257.83mW, P_{Rx}=123.74mW.



(a) Average total energy saved. (b) Average total energy consumption. (c) Average total communication energy.

Fig. 13: Average total energy versus uplink and downlink rates in WiFi for the apps with Layer-by-Layer CDG ($s = 5$) and 14 components while T=3s, P_{Tx}=257.83mW, P_{Rx}=123.74mW.

through cloud offloading increases (Fig. 13a), and the total energy consumption decreases (Fig. 13b). Moreover, we see that while the probability and rates increase, the energy saved through remote execution decreases.

Figures 14 and 15 show the performance of the proposed JSCO scheme for randomly generated Fan-in/Fan-out CDGs. In Fig. 14, energy saved, energy consumed by the mobile, and energy consumed for communication are respectively plotted versus the average data size for each transfer. Our results indicate that the performance of the JSCO scheme is independent of the Fan-in/Fan-out ratio of these graphs but dependent on the total Fan-in plus Fan-out degrees. When the in+out degree increases, the dependency and offloading costs increase such that the energy saved through cloud offloading decreases and the energy consumed by the mobile device increases (Figures 14a, 14b). Also when the data size for transferring increases, the energy consumed by the mobile increases (Fig. 14c). Fig. 15 presents the energy as a function of the uplink/downlink rates. While rates increase and the in+out degree decreases, the energy consumed for communication decreases (Fig. 15c). Therefore, the energy saved through remote execution increases (Fig. 15a), and the energy consumed by the mobile decreases (Fig. 15c).

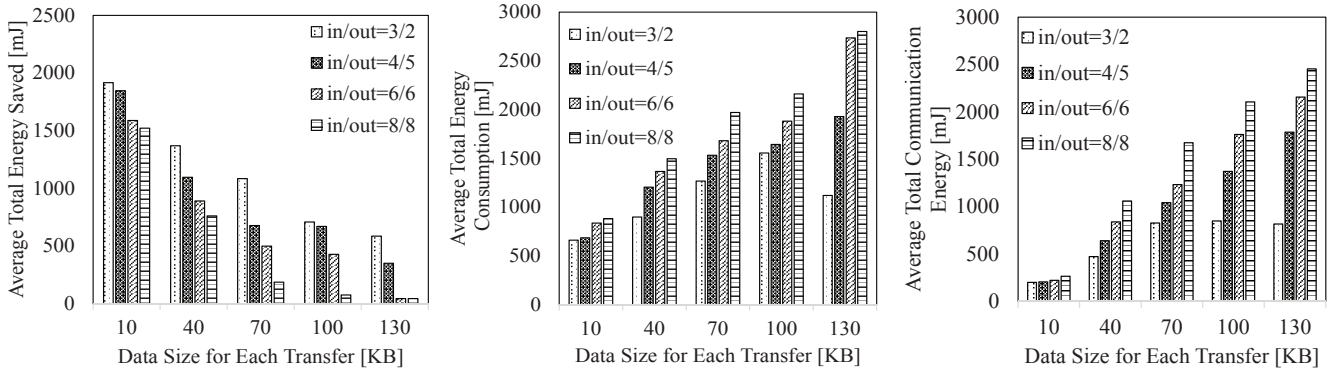
5.5 Scalability of the JSCO Scheme

TABLE 2: Program runtimes of the CPLEX optimizer for the proposed LP using Layer-by-Layer and Fan-in/Fan-out CDGs.

N	mobile-only execution time [ms]	T [ms]	runtime for Layer-by-Layer [s]	runtime for Fan-in/Fan-out [s]
25	7714	6500	561	439
45	16230	13500	924	834
65	17412	14250	1764	1649
85	27877	17100	2862	2700
105	28098	21000	7654	8647

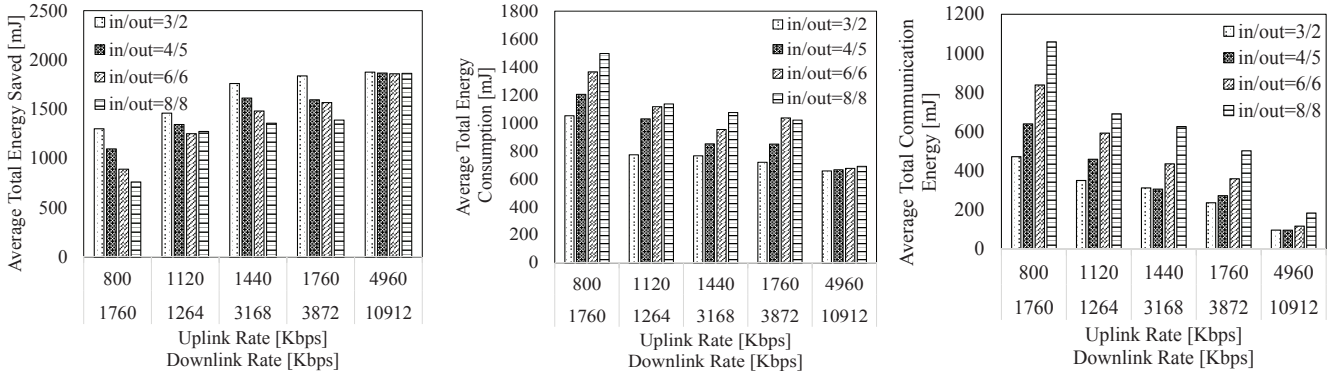
In this subsection, we discuss the scalability of our JSCO scheme. Specifically, we want to address the largest application that the JSCO scheme can handle in terms of the number of components and total execution time. In our discussions so far, we have used only a 14-component application (either real or randomly generated). In order to maintain the same probability distribution of our measurements when scaling up the application, we calculate the histogram of the current real data measurements (q_k^{m}, q_k^{c}, P_{ac} $\forall k$) from the 14-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560808, IEEE Transactions on Cloud Computing



(a) Average Total energy saved. (b) Average total energy consumption. (c) Average total communication energy.

Fig. 14: Average total energy versus required data size for transferring each component in the apps with Fan-in/Fan-out CDGs and 14 components while T=3s, P_{Tx}=257.83mW, P_{Rx}=123.74mW.



(a) Average Total energy saved. (b) Average total energy consumption. (c) Average total communication energy.

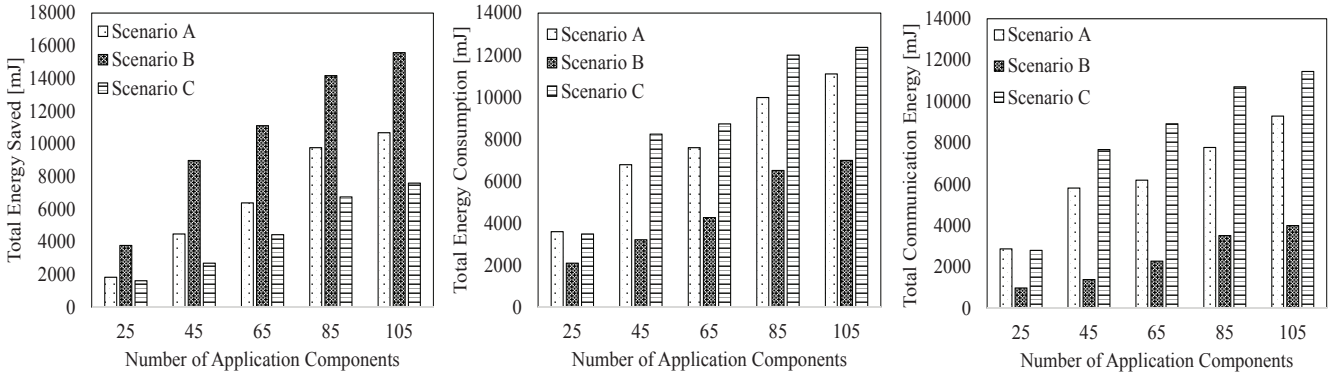
Fig. 15: Average total energy versus uplink and downlink rates in WiFi for the apps with Fan-in/Fan-out CDGs and 14 components while T=3s, P_{Tx}=257.83mW, P_{Rx}=123.74mW.

component video navigation application. Using the obtained distribution, we generate the new data for applications with a greater number of components (25, 45, 65, 85, and 105 components). Increasing the number of components requires a corresponding increase in the runtime deadline (T); for example, for a 25-component application $T = 6500$ms; for $N = 45$, $T = 13500$ms; for $N = 65$, $T = 14250$ms; for $N = 85$, $T = 17100$ms; and for $N = 105$, $T = 21000$ms.

Table 2 shows the program runtimes using the proposed scheme for the two types of randomly generated CDGs– Layer-by-Layer and Fan-in/Fan-out. In this table, we consider the total execution time in accordance with the number of components (N). We see that while the number of components and total execution time increase, the runtime of the proposed scheme increases. The JSCO scheme is capable of handling over 100 components with a mobile-only execution time of 28 seconds. Our simulations were done on a single server machine with an Intel Xeon(R) E7340 processor @ 2.5GHz CPU and 60GB of RAM. Although the runtime to solve the associated integer linear program increases with the number of components to over 2hours, this time can be reduced through parallel implementation using more powerful processors.

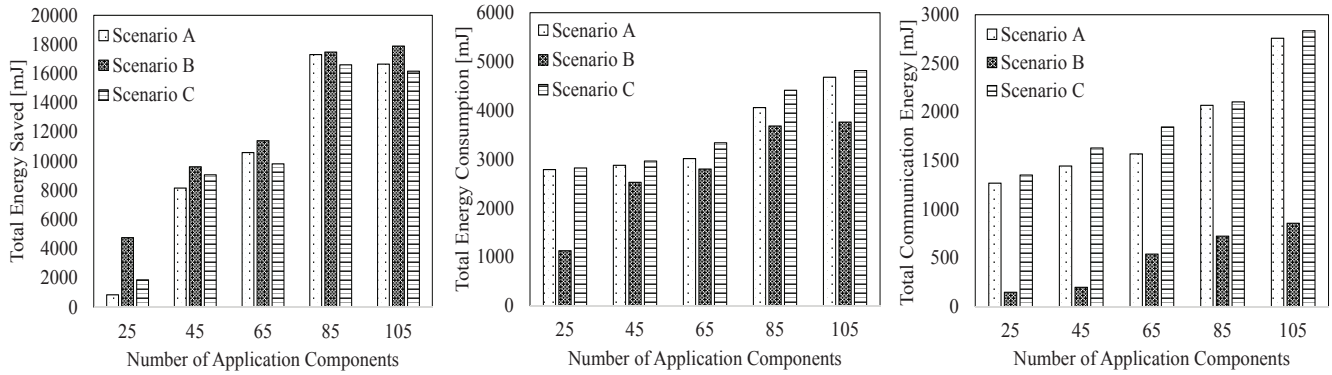
Three scenarios are considered in this part: (*A*) the scenario where the average data size to transfer is fixed at 1220KB and the uplink/downlink rate is fixed at 1.28/2.816Mbps; (*B*) the scenario where the average data size to transfer is fixed at 1220KB (the same as *A*) and the uplink/downlink rate is fixed at 4.96/10.912 Mbps (more than *A*); and (*C*) the scenario where the average data size to transfer is fixed at 2196KB (more than *A*) and the uplink/downlink rate is fixed at 1.28/2.816Mbps (the same as *A*). In Fig. 16, the Layer-by-Layer CDG with a larger number of components is considered. In this figure, the energy saved, total energy consumed, and energy consumed for communication are respectively shown as a function of the number of application components for the three scenarios, *A*, *B*, *C*. Note that here, $p = 0.2$ and $s = 5$ (which is the number of layers). When the number of application components increases, the edges between components increase and therefore the costs of offloading increase. Thus, all the energy values increase. We can see that while the rates increase in Scenario *B* in comparison to Scenario *A*, the energy saved through remote execution increases, energy consumed for offloading decreases, and the total energy consumed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560808, IEEE Transactions on Cloud Computing



(a) Total energy saved. (b) Total energy consumption. (c) Total communication energy.

Fig. 16: Total energy versus the number of application components with Layer-by-Layer CDG (p=0.2 and $s = 5$), presented in Scenarios A, B, and C.



(a) Total energy saved. (b) Total energy consumption. (c) Total communication energy.

Fig. 17: Total energy versus the number of application components with Fan-in/Fan-out CDGs (maximum in-degree is 3 and maximum out-degree is 2), presented in Scenarios A, B, and C.

by the mobile device decreases. On the other hand, while the data size increases in Scenario *C* in comparison to Scenario *A*, the energy saved decreases, communication energy increases, and the total energy consumed by the mobile device also increases, which is all as expected.

Fig. 17 plots the energy values in accordance with the number of application components for the applications with Fan-in/Fan-out CDGs in the three scenarios, *A*, *B*, *C*. Here maximum in-degree is set to 3 and the maximum out-degree is set to 2 for the corresponding CDGs. Similar observations as in Fig. 16 are made here as well.

6 Conclusion and Future Work

In this paper, we proposed *the first* energy-efficient JSCO scheme for mobile devices using applications with arbitrary component dependency graphs. Existing work considers either sequential ordering of the components or a pre-determined ordering, leading to less adaptability with wireless conditions. This was cast as an optimization problem, and the results using real data measurements show that the proposed JSCO reduces consumption by 54% compared to local execution and up to 37% compared to other existing schemes. Future work includes devising polynomial-time heuristics to reduce the runtime for the optimization problem.

References

[1] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[2] K. Kumar and Y. H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, pp. 51–56, Apr. 2010.

[3] N. Vallina-Rodriguez and J. Crowcroft, "Energy management techniques in modern mobile handsets," *IEEE Communications Surveys Tutorials*, vol. 15, no. 1, pp. 179–198, First 2013.

[4] S. E. Mahmoodi, K. P. Subbalakshmi, and V. Sagar, "Cloud offloading for multi-radio enabled mobile devices," in *IEEE International Communication Conference (ICC)*, Jun. 2015, pp. 1–6.

[5] X. Chen, J. Wu, Y. Cai, H. Zhang, and T. Chen, "Energy-efficiency oriented traffic offloading in wireless networks: A brief survey and a learning approach for heterogeneous cellular networks," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 4, pp. 627–640, Apr. 2015.

[6] X. Ma, Y. Zhao, L. Zhang, H. Wang, and L. Peng, "When mobile terminals meet the cloud: computation offloading as the bridge," *IEEE Magazine on Network*, vol. 27, no. 5, pp. 28–33, Sep. 2013.

[7] B. Zhou, A. Dastjerdi, R. Calheiros, S. Srirama, and R. Buyya, "A context sensitive offloading scheme for mobile cloud computing service," in *IEEE International Conference on Cloud Computing (CLOUD)*, Jun. 2015, pp. 869–876.

[8] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-Based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[9] P. Shu, F. Liu, H. Jin, M. Chen, F. Wen, and Y. Qu, "eTime: Energy-efficient transmission between cloud and mobile devices," in *IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2013, pp. 195–199.
[10] Y.-D. Lin, E.-H. Chu, Y.-C. Lai, and T.-J. Huang, "Time-and-Energy-Aware computation offloading in handheld devices to coprocessors and clouds," *IEEE Systems Journal*, vol. 9, no. 2, pp. 393–405, Jun. 2015.
[11] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Transactions on Wireless Communications*, vol. 12, no. 9, pp. 4569–4581, Sep. 2013.
[12] H. Wu, Q. Wang, and K. Wolter, "Trade-off between performance improvement and energy saving in mobile cloud offloading systems," in *IEEE International Conference on Communications Workshops (ICC)*, Jun. 2013, pp. 728–732.
[13] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," in *Proceedings of the International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys. ACM, 2010, pp. 49–62.
[14] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *IEEE proceedings of INFOCOM*, 2012, pp. 945–953.
[15] W. Zhang, Y. Wen, and D. Wu, "Collaborative task execution in mobile cloud computing under a stochastic wireless channel," *IEEE Transactions on Wireless Communications*, vol. 14, no. 1, pp. 81–93, Jan. 2015.
[16] S. Barbarossa, S. Sardellitti, and P. Di Lorenzo, "Computation offloading for mobile cloud computing based on wide cross-layer optimization," in *Future Network and Mobile Summit (FutureNetworkSummit)*, Jul. 2013, pp. 1–10.
[17] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Transactions on Wireless Communications*, vol. 11, no. 6, pp. 1991–1995, Jun. 2012.
[18] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya, "Mobile code offloading: from concept to practice and beyond," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 80–88, Mar. 2015.
[19] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the conference on Computer systems*, 2011, pp. 301–314.
[20] [Online]. Available: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.
[21] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J.-M. Vincent, and F. Wagner, "Random graph generation for scheduling simulations," in *Proceedings of the International ICST Conference on Simulation Tools and Techniques*, 2010, pp. 60:1–60:10.
[22] P. Balakrishnan and C. K. Tham, "Energy-efficient mapping and scheduling of task interaction graphs for code offloading in mobile cloud computing," in *IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, Dec. 2013, pp. 34–41.
[23] D. Kovachev, T. Yu, and R. Klamma, "Adaptive computation offloading from mobile devices into the cloud," in *IEEE International Symposium on Parallel and Distributed Processing with Applications*, Apr. 2012, pp. 784–791.
[24] M. Nir, A. Matrawy, and M. St-Hilaire, "An energy optimizing scheduler for mobile cloud computing environments," in *IEEE Conference on Computer Communications Workshops (INFOCOM workshops)*, Apr. 2014, pp. 404–409.
[25] [Online]. Available: http://www.opengl.org/.
[26] [Online]. Available: http://www.developer.com/ws/android/programming/face-detection-with-android-apis.html.
[27] [Online]. Available: http://opencv.org/.
[28] P. Rubin. [Online]. Available: http://orinanobworld.blogspot.de/2010/10/binary-variables-and-quadratic-terms.html.
[29] [Online]. Available: http://www-01.ibm.com/support/docview.wss?uid=swg21399933.
[30] [Online]. Available: http://darnok.org/programming/face-recognition/.
[31] [Online]. Available: http://code.google.com/p/currentwidget/.
[32] S. Ou, K. Yang, and J. Zhang, "An effective offloading middleware for pervasive services on mobile devices," *Pervasive and Mobile Computing*, vol. 3, no. 4, pp. 362 – 385, Aug. 2007.
[33] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, Mar. 2002.
[34] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 175–186, Mar. 2015.
[35] [Online]. Available: http://www.3gpp.org/ftp/tsg-ran/wg4-radio/.
[36] U. Kremer, J. Hicks, and J. M. Rehg, "Compiler-directed remote task execution for power management," in *Workshop on Compilers and Operating Systems for Low Power*, Oct. 2000.

S. Eman Mahmoodi is currently pursuing his PhD degree at the Department of Electrical and Computer Engineering, Stevens Institute of Technology. He received the BS and MS degree in Electrical Engineering from Iran University of Science and Technology, respectively in 2009 and 2012. He has been working on Mobile Cloud Computing, Optimization and Applied Modeling, Cognitive Networks, and Wireless Communications. Mahmoodi is a Stevens Innovation and Entrepreneurship Doctoral Fellow.

R. N. Uma 's research interests include data science, scheduling and resource allocation with applications to cloud computing, robotics, wireless sensor networks, multimedia networking, and large logistics problems. She received her BSc degree in Mathematics from the University of Madras, Chennai, India, the ME degree in Computer Science from the Indian Institute of Science, Bangalore, India, and the PhD degree in Computer Science from the NYU Tandon School of Engineering (formerly, Polytechnic University) New York. She is an associate professor in the Department of Mathematics and Physics at North Carolina Central University, Durham. She is a member of the IEEE and the ACM.

K. P. (Suba) Subbalakshmi is a Professor at Stevens Institute of Technology, and will serve as a Jefferson Science Fellow in 2016. Her research interests span: Cognitive radio networks, Cognitive Mobile Cloud Computing, Social Media Analytics and Wireless security. She is a Founding Associate Editor of the IEEE Transactions on Cognitive Communications and Networking and an Associate Editor of the IEEE Transactions on Vehicular Technology. She is the Founding Chair of the Security Special Interest Group of the IEEE Technical Committee on Cognitive Networks. She is also a recipient of the NJIHOF Innovator award.