Distributed Computational Load Balancing for Real-Time Applications

Saurav Sthapit, James R. Hopgood, and John Thompson
Institute of Digital Communications, School of Engineering, University of Edinburgh
Emails:{s.sthapit, james.hopgood, john.thompson}@ed.ac.uk

***Abstract*—Mobile Cloud Computing or Fog computing refer to offloading computationally intensive algorithms from a mobile device to a cloud or a intermediate cloud in order to save resources (time and energy) in the mobile device. In this paper, we look at alternative solution when the cloud or fog is not available. We modelled sensors using network of queues and use linear programming to make scheduling decisions. We then propose novel algorithms which can improve efficiency of the overall system. Results show significant performance improvement at the cost of using some extra energy. Particularly, when incoming job rate is higher, we found our Proactive Centralised gives the best compromise between performance and energy whereas Reactive Distributed is more effective when job rate is lower.**

***Index Terms*—Offloading, Mobile Cloud Computing, Energy, IOT, Fog Computing, Edge Computing**

I. Introduction

Usage of *commercial off-the-shelf* (COTS) smart devices in defence and surveillance applications is an interesting prospect. As an example application, imagine a swarm of COTS drones flying and gathering visual intelligence on a missing person or an armed terrorist (See Fig. 1). Reporting raw data back to a base station is prohibitive in terms of both time and energy. Even worse, if it is a covert defence operation, it may open up the base to external attacks. So some pre-processing must be done on the drone itself; for example only report to the base once the individual is recognised. For that, the drones must be able to run *person re-identification* (PRID) algorithms for the targets appearing in its Field Of View (FOV). The time complexity of the PRID algorithms is substantially higher than other algorithms running in the algorithm chain [1]. The drones may have different computing and energy resources and depending on the state of the device, it may not be able to complete these processing in an allocated time. Traditional Mobile Cloud Computing (MCC) in which jobs is outsourced to the cloud may not be available or feasible depending on the communication channel to the cloud [2]. Recently, Fog/Edge computing has been introduced whereby mobile devices offload nearby servers (preferably at base stations) instead of cloud (see [3]). However, Fog computing could be unavailable just like the cloud. (For example in underground or battlefield far from the base station).

In this paper, we propose algorithms to balance the computational load among the smart cameras for soft real-time applications. For rest of this paper, we consider a network of smartphones trying to run PRID algorithms as our exemplar problem and make the assumption listed below. However, the algorithms can be generalised to other problems such as multistatic radar or sonar, distributed audio processing etc.

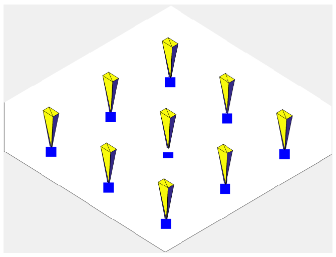
Fig. 1: Nine Camera sensors and their FOV. Blue square represents camera in a drone or a smart phone

1) In a network of cameras, targets are spatially and temporally distributed. That means, more targets may appear in some cameras than others and at different times. So, nodes may be able to help each other.
2) The jobs arriving at the node can be *offloadable* or *non-offloadable* depending whether the *offloader* can save *time* or *energy* by offloading the job to others [4].
3) As long as the total job rates (across all nodes) is less than the total computing capability of the network of nodes, it should be possible to trade energy with performance and productivity.

The problem we are trying to tackle is two fold. First, we want to make a scheduling decision for *offloadable* jobs among the nodes. Second, we need to determine the Node State Information (NSI) that needs to be shared, and the frequency, in order to make the scheduling decision. Queuing theory abstracts our scheduling algorithms of the underlying hardware. It means the system may consist of Central Processing Unit (CPU) nodes or dedicated accelerators such as Graphical Processing Unit and Field Programmable Gate Array and also, we avoid the need to take the decision for each and every tasks. This work extends our previous work [2], where sensors take decision using simple cost functions and on task by task basis. Also the scenario is changed as in this work we do not consider cloud at all. Wu et. al [5] have used queuing theory approach for MCC but their focus is on offloading to the cloud and availability of communication channels. In this paper, we use linear programming to make the scheduling decision. Then we propose a *purely distributed* solution where nodes only need to communicate to neighbouring nodes directly connected to them. Based on where the solver is executed and how data is shared, we propose four novel algorithms and compare their performance with the non-offloading case. In summary, the

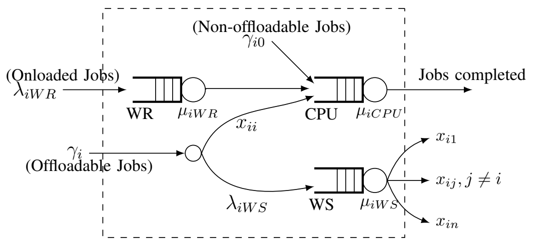

Fig. 2: A sensor node modelled as network of queues. CPU, WR, WS represent CPU, WiFi Receiver and WiFi Sender queues respectively

main contributions of this paper are as follows:

- Propose novel algorithms for on-line workload balancing for real-time applications in distributed systems.
- Propose Offloading Cost function that incorporates NSI such as battery level, bandwidth and CPU availability.
- Show the proposed algorithms improve the performance of the overall network of battery powered sensor system compared to Non Offloading (NO) system.

In the next section, we model the node network using a *network of queues* and formally define the problem. Section III details the proposed algorithms. In section IV, we describe the experimental settings and the results of the simulations. Finally we discuss and conclude our findings in section V.

II. System Model

Let us model a network of sensors depicted in Fig. 1. Following the notation in [6], let $G = (N, A)$ be a directed network defined by a set N of n nodes and a set A of m directed arcs. Each arc $(i, j) \in A$ represents a communication link (for example WiFi) from node i to j, and has an associated cost c_{ij} that denotes cost per unit flow on that arc.

A. Node

Each node i is a smartphone or a similar device with a CPU, WiFi, cellular link and a camera. We use $M/M/1$ queues to model behaviour of each of these components. Specifically, the $M/M/1$ has First Come First Service (FCFS) scheduling discipline, an arrival process that is Poisson and service time that is exponentially distributed [7]. Similarly, for the communication part, we model WiFi using two $M/M/1$ queues (sender and receiver side). We assume a common WiFi send and receive rate (i.e $\mu_{iWS} = \mu_{iWR} = \mu_{iWF}$). The resulting model of the node is depicted in Fig. 2. Each node i can be defined as a tuple $\{\gamma_i, \gamma_{i0}, \mu_i, \mu_{iWF}\}$ where γ_i is the rate of *offloadable jobs*, γ_{i0} is the rate of *non-offloadable jobs*, μ_{iCPU} is the service rate of CPU, μ_{iWF} is the WiFi transmission rate. We define this node information as Node State Information (NSI). Each individual target that passes through a camera's FOV generates an *offloadable job*. Jobs that are integral to the node itself such as operating system load and algorithms which do not benefit from offloading are termed as *non-offloadable jobs*. They may be spatially and temporarily distributed as well like the *offloadable jobs*.

B. Network of Queues

A network of queues is defined as an open network if there are external jobs coming into the system. Such networks can be modelled using the Open Jackson network [7]. Vilaplana [8] used it for modelling cloud computing paradigm. The Open Jackson network states that the arrival rate for a queue $a \in \{1, ..., k\}$ is given by Eqn.(1). Based on this formulation, we can calculate the incoming and outgoing job rates of all the queues in our system.

$$\lambda_a = \gamma_a + \sum_{b=1}^{k} p_{ba}\lambda_b \tag{1}$$

where,

γ_a is the rate of arrival of external targets

λ_b is the arrival rate at queue b,

p_{ba} is the prob. a job but moves from queue b to queue a

C. Problem Formulation

We formulate the scheduling decision problem as a minimum cost flow problem (Eqn. (2)) with constraints that all the jobs get scheduled and without compromising the stability of the queues. The decision variable $x_{ij} \in R^{(n \times m)}$ represent the job flow on an communication link $(i, j) \in A$. x_{ii} is the job rate that is executed locally. We can guarantee the rate stability of a queue by ensuring the average arrival rate is less than the average service rate. Hence, if the average incoming job rate for the CPU queue in a node is greater than its service rate, we should look for alternatives. The equality constraint in (2b) makes sure that all the jobs are assigned whereas the inequality constraint in (2c) makes sure that the jobs can be processed by corresponding nodes they are assigned. This formulation uses NSI from all the nodes (n) and makes decision for all the nodes simultaneously. The cost function for the problem is described in the section II-E.

$$X = \underset{x}{\operatorname{argmin}} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{2a}$$

$$\text{subject to (s.t.)} \sum_{j=1}^{n} x_{ij} = \gamma_i, \qquad \forall i \in N \tag{2b}$$

$$\sum_{j=1}^{n} x_{ji} + \gamma_{i0} \preceq \mu_{iCPU}, \qquad \forall i \in N \tag{2c}$$

$$x_{ij} \geq 0 \tag{2d}$$

The solution of Eqn. (2) can be rewritten as a decision matrix shown below:

$$X = \begin{bmatrix} x_{11} & . & x_{1i} & . & x_{1n} \\ . & . & . & . & . \\ x_{i1} & . & x_{ii} & . & .x_{in} \\ . & . & . & . & . \\ x_{n1} & . & x_{ni} & . & x_{nn} \end{bmatrix} \tag{3}$$

decision vector: We define each row of X as a *decision vector*(dv). The dv_i tells node i how it should process the incoming targets. Also, it is seen easily that i^{th} column of the matrix indicates how other nodes are offloading to i^{th} node.

D. Distributed solution

Given the time varying nature of the job arrival rate, we need to solve the problem in Eqn. (2) frequently. Each node only cares about its own *column* and *row* of the decision matrix X. We later see in section II-F that the complexity of the problem depends on n and m which is the total number of nodes and arcs respectively. So we simplify the problem by *primal decomposition* whereby each node calculates its own dv. This is similar to the *Gauss-Siedel* like method used by Meskar [9] for MCC. The algorithm basically communicates with its immediate neighbours to see what they can offer and takes the decision. The approach is not selfish as it considers neighbours' resources rather than offloading everything. The problem is defined below for each node $i \in N$. It is different from the central problem in Eqn.(2) as that each node i only tries to minimise the cost of its own objective function on the basis of information available on its immediate neighbours.

$$\mathrm{dv}_i = \operatorname*{argmin}_x \sum_{j=1}^{n} c_{ij} x_{ij} \tag{4a}$$

$$\text{s.t.} \sum_{j=1}^{n} x_{ij} = \gamma_i; \sum_{i=1}^{n} x_{ji} + \gamma_{i0} \preceq \mu_{iCPU}; x_{ij} \geq 0 \tag{4b}$$

E. Cost function

Once we are certain that all the arriving jobs can be scheduled such that the queues are all rate stable, we would like to achieve it with the minimum cost. We define the cost function c_{ij} as the cost of scheduling a unit job from node i to node j as shown below.

$$c_{ij} = \begin{cases} \omega_3 L_i, & \text{if } i = j \\ \frac{\omega_1 D(f+1)}{BW_{ij}} + \frac{\omega_2 B_i}{B_j} + \omega_3 L_j, & \text{if } i \neq j, (i,j) \in A \\ \infty, & \text{if } i \neq j, (i,j) \notin A \end{cases} \tag{5}$$

where, D is the data size

f is the average retransmission times (see Eqn. 6a)

BW_{ij} is the expected bandwidth between node i and j

B_i, B_j are the remaining battery in node i, j

L_i, L_j is the number of jobs already in node i, j

$\omega_1, \omega_2, \omega_3$ are weight factors

The cost comprises of three distinct components; the communication cost and the remaining battery level and the availability of CPU. Their significance can be changed using the weighting factor ω_1, ω_2 and ω_3.

1) Communication cost: The communication cost depends on the expected bandwidth between two nodes, data size and a retransmission factor f. As the communication channel is not perfect thanks to various noise and interference, we account them using the retransmission factor f. In the experiments, we randomly sample Packet Delivery Rate (PDR) between two nodes and use mean of the geometric distribution to calculate the average number of transmissions to send the data from one node to another (see Eqn. 6a). The relationship (see Fig. 3a) shows us that as the PDR degrades, average number of retransmission rises exponentially. For the simulations, we consider 0.5 as the minimum PDR for any valid communication link.

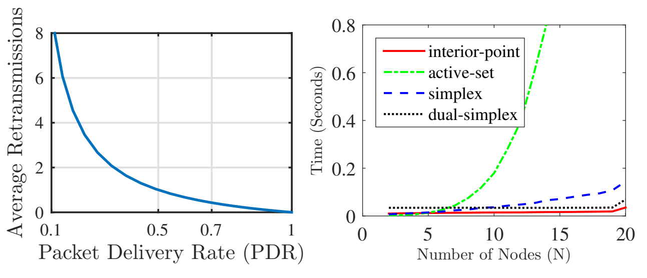

Fig. 3: (a) Average no. of retransmissions required due to imperfect channel. (b) Time complexity of various linear problem solvers

$$f(\mathrm{PDR}) = \mathbb{E}[g(x; \mathrm{PDR})], \text{ where} \tag{6a}$$

$$g(x; \mathrm{PDR}) = \mathrm{PDR}(1 - \mathrm{PDR})^{x-1}, \forall x \in \{0, .., \infty\} \tag{6b}$$

2) Energy available: The second element of our cost function is the ratio of battery level of the nodes.

3) CPU availability: We use number of existing jobs in the CPU queues as the measure of CPU availability. Higher number suggest low availabity and vice versa. This is also applicable for self-processing in the scheduling decision making.

F. Computational Complexity

The optimisation problem stated in Eqn. (2, 4) can be solved using efficient linear programming techniques. Dual Simplex and Interior Point algorithms are popular methods of solving linear problems. Interior point algorithms are considered to be efficient and also require less memory than others. We performed experiments to gauge their *time complexity* for different number of nodes and found interior point to be the most efficient (see Fig. 3b). These experiments were performed on a desktop computer with an Intel Xeon processor and running MATLAB 2015a under linux environment. The runtime of these algorithms on a embedded device may be significantly higher but should follow the similar pattern.

III. Algorithms

In the previous section, we formulated the problem of scheduling jobs as *central* and *distributed* problems. We have selected a co-operative environment in which all the nodes tries to achieve global objectives (i.e. process most jobs in an allocated time). By co-operative, we mean if a node sends a job to another node, the other node must execute it. However, we consider the nodes are not selfish and only offloads if required. We consider two data sharing mechanism; *proactive* and *reactive*. As we will see in Section IV, *proactive* is suitable when incoming job rate is *high* and decisions have to be made often whereas the *reactive* is more suited to quite environments. So depending on how this data is shared amongst the nodes and where the algorithms is run we propose following four algorithms. We compare all four algorithms against the NO case when we do not allow offloading at all.

A. Oracle (O)

The *Oracle* has access to all the sensor node's NSI at all times. The *Oracle* solves the cost minimization problem in Eqn. (2) every second and sends dv to all nodes simultaneously. It is not feasible in practice but gives the best result for comparison. Experiments show that even ignoring the cost of communication of NSI and cost of executing the solver, it consumes the most energy.

B. Proactive Centralised (PC)

This is a more realistic version of the *Oracle*. In this method all the n nodes send NSI to a *nominated server* which then solves Eqn.(2) and sends corresponding dv back to each nodes. In simulation, the *server* has connection to all the nodes but this isn't necessary as NSI and dv can be conveyed using multiple hops. We consider the cost of communication of NSI as well as cost of executing the solver. All other nodes are obliged to follow the decision made by the *server* and computes and offloads based on the dv until a new one is broadcast. We want to investigate this case to find out how often we can broadcast the NSI without using too much communication resources Obviously, there isn't a single answer but it depends on many factors such as the communication bandwidth, size of NSI, PDR and number of nodes in the set. If there are $n-1$ nodes sending their NSI to the *server* every t seconds, the queue with the highest probability of being busy is the *server*'s receiving queue. We analyse its performance below.

$$\text{Arriving rate } (\lambda) = \frac{n-1}{t} \tag{7}$$

$$\text{Worst Service rate } (\mu) = \frac{\text{Data Rate} \times \text{worst PDR}}{\text{NSI size}} \tag{8}$$

$$\text{Utilization } (\rho) = \frac{\lambda}{\mu} = \frac{(n-1) \times \text{ NSI size}}{t \times \text{Data Rate} \times \text{PDR}} \tag{9}$$

$$P[0] = 1 - \rho$$

where, $P[0]$ is the probability there is no jobs in the queue

Based on the arriving rate and service rate we can estimate the peformance of *server*'s receive queue. For example, say there are 11 sensors connected with a data rate of 54 Mbps, PDR of 0.7 and NSI of 1 Mbits, send NSI every 10 seconds. Then Eqn. (6a) estimates the queue utilisation is ≈ 0.03 and no waiting times for $\approx 97\%$ of the time. Similarly the average delay is around ≈ 0.03 seconds. Fig. 4 shows waiting times at the receiving node at various intervals and for different speeds.

C. Proactive Distributed (PD)

PD is similar to PC except for three main differences.

1) It is purely distributed. There is no *server* and each node has to solve its own optimisation problem.
2) Instead of solving central problem in Eqn.(2), each node only solves distributed problem in Eqn.(4).
3) Set N contains immediate rather than neighbours than all the nodes. Even if total nodes is large (> 100), we N may be limited to tens of nodes.

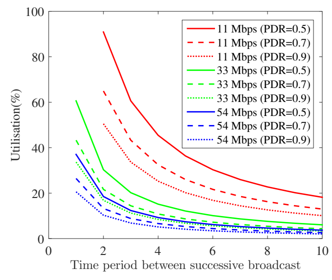

Fig. 4: Queue utilisation of the *server* in proactive under various network conditions and NSI update frequency. NSI size set to 1 Mb.

D. Reactive Distributed (RD)

If only a few nodes get overloaded and infrequently, transmitting NSI regularly can be a waste of energy. Also, tail-end behaviour User Equipment (UE) may mean regular transmission forces UE to stay in the high powered state instead of the low powered idle state [10]. In this method (see Alg. (1)), nodes only communicate when they need help. The node seeking help broadcasts Request For Help (RFH) and waits until the neighbours respond by sending their NSI. Neighbouring nodes must respond if their average CPU usage is less than a threshold. Once the node seeking help receives NSI from other nodes, it formulates and solves Eqn. (4). To avoid using old information and update neighbour's current situation, we also set a timer T_{th} after which the node has to start again by broadcasting the RFH.

Algorithm 1 Reactive Distributed

```
if $\gamma_i + \gamma_i 0 \leq \mu_i$ then
    Set dv$_i$ to not offload.
else
    if RFH broadcasted & decision time $< T_{th}$ then
        Follow previous dv$_i$
    else
        Broadcast RFH to all nodes.
        Wait $T_{wait}$ seconds for NSI
        if No of NSI received $\geq 2$ then
            Solve Eqn.(4) for new dv$_i$ and follow it.
        else
            Broadcast RFH again, follow previous dv$_i$.
        end if
    end if
end if
```

IV. Simulator and Experimental Results

We use the simulator [2] which uses a utilisation based model by Jung et. al [10] and their parameters for Google Nexus I phone to estimate the energy consumption of the nodes. The simulator has evolved to accommodate targets moving in three dimensions (such as *drones*). The simulator is set up to simulate nine smartphones placed on a 3×3 grid as shown in Fig. 1. For this paper, the exact number and the configuration is chosen empirically. In future, different setting will be explored. Each blue square representing a smartphone can detect targets passing through its FOV represented by blue/yellow cone shape in the figure. For

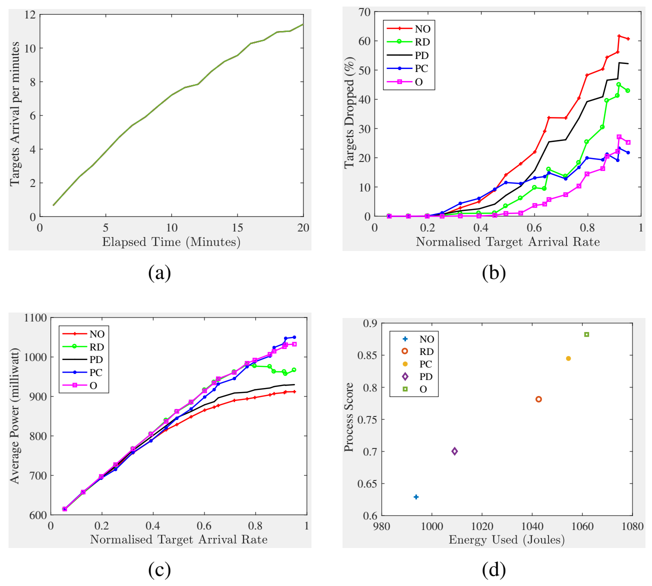
Fig. 5: (a) Target arrival rate per nodes over simulation time. (b) Targets dropped over Arrival Rate. (c) Power Consumed over Arrival Rate (d) Efficiency Score of proposed algorithms

target simulation, we used Random Waypoint Model (RWP) [11]. In RWP, targets spawn at random locations in a three-dimensional space. The targets either pause for certain time or select its next destination. When it selects its next destination it moves towards it with a random but constant velocity; the process repeats until it moves out of the platform. A non-uniform spatial phenomenon of the RWP means that targets are concentrated in the middle of the platform [11]. We use this phenomenon and irregular FOV to simulate irregular loads among the nine sensors. Sensor 5, which is in the middle of the platform detects the highest number of targets.

We ran 100 Monte-Carlo simulations for 20 minutes of simulated time. The target spawning rate is higher than dying rate, so target rate generally increases over time across all nodes (see Fig. 5a). Every minute we take a snapshot of targets dropped and energy consumed and plot it as a function of target arrival rate (γ)(see Fig.5b, 5c). As expected, Oracle gives the best results whereas NO is the worst performer. PC gives the next best results, however also consumes more energy. Upto 60% of the total normalised arrival rate, RD and PD performs better than PC and significantly better than the non-offloading case. Yet the power consumption of RD is just marginally higher than PC and PD is even lower than PC around that point. However, the performance of distributed algorithms significantly degrades as the target arrival rate goes up. Also, Fig. 5c also shows lower power consumption at higher target arrival rate for RD which also coincides with its fall in performance. This is due to more neighbours being busier. It shows that *distributed* algorithms may be best suited to lower arrival rates whereas the centralised approach is suited to the higher job arrival rates.

Next we define *process score* as the percentage of jobs

TABLE I: Simulation Results (Averaged over 100 runs)

Algo-rithm	Arrival Rate (/min)	Service Rate (/min)	Process Score	Energy Used (Joules)
NO	6.91	4.35	0.63	994
RD	6.91	5.34	0.78	1043
PD	6.91	4.84	0.70	1009
PC	6.91	5.84	0.85	1055
O	6.91	6.09	0.88	1062

successfully executed in the allocated times and *efficiency score* as the ratio of *Successful Executions* to the energy consumed [2]. We summarise the overall results in Table I and Fig. 5d. There is almost a linear relationship between performance and energy consumption in Fig.5d meaning performance can be enhanced by spending extra energy.

V. Conclusion

In this paper, we modelled sensor network as network of queues using Open Jackson network. We proposed various *reactive* and *proactive* algorithms which significantly enhanced the performance of the system compared to the NO scenario. The results reinforces our belief that we can process all the jobs if, the total job rate is less than total computing capability, and if other node's NSI is available. Also depending on normalised job arrival rate, *reactive distributed* or *proactive centralised* may be more suited. It is possible to formulate a hybrid strategy, which can switch between them based on the job arrival rate. In future work, we plan to run our algorithms on real dataset and dynamic scenarios.

VI. Acknowledgement

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) [EP/K014277/1] and the MOD University Defence Research Centre (UDRC) on Signal Processing.

References

[1] S. Sthapit, J. Thompson, J. Hopgood, and N. Robertson, "Distributed Implementation for Person," in *Sens. Signal Process. Def.*, 2015.
[2] S. Sthapit, J. R. Hopgood, N. M. Robertson, and J. Thompson, "Offloading to Neighbouring Nodes in Smart Camera Network," in *EUSIPCO*, 2016.
[3] S. Yi, C. Li, and Q. Li, "A Survey of Fog Computing: Concepts, Applications and Issues," *Proc. 2015 Work. Mob. Big Data - Mobidata '15*, pp. 37–42, 2015.
[4] K. Kumar and Y.-H. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?," *Comput. (Long. Beach. Calif).*, vol. 43, no. 4, pp. 51–56, 2010.
[5] H. Wu, W. Knottenbelt, and K. Wolter, "Analysis of the Energy-Response Time Tradeoff for Delayed Mobile Cloud Offloading," in *SIGMETRICS Perform. Eval. Rev.*, vol. 43, pp. 33–35, 2015.
[6] T. L. M. Ravindra K. Ahuja, *Network Flows: Theory, Algorithms, and Applications*, vol. 1. 1993.
[7] W. J. Stewart, *Probability, Markov chains, queues, and simulation : the mathematical basis of performance modeling*. Princeton (N.J.), Oxford: Princeton University Press, 2009.
[8] J. Vilaplana, F. Solsona, I. Teixidó, J. Mateo, F. Abella, and J. Rius, "A queuing theory model for cloud computing," *J. Supercomput.*, vol. 69, pp. 492–507, 2014.
[9] E. Meskar, T. D. Todd, D. Zhao, and G. Karakostas, "Energy Aware Offloading for Competing Users on a Shared Communication Channel," *IEEE Trans. Mob. Comput.*, vol. 16, no. 1, 2017.
[10] W. Jung, C. Kang, C. Yoon, D. D. Kim, and H. Cha, "DevScope: A Nonintrusive and Online Power Analysis Tool for Smartphone Hardware Components," *Proc. Eighth IEEE/ACM/IFIP Int. Conf. Hardware/Software Codesign Syst. Synth.*, pp. 353–362, 2012.
[11] F. Bai and A. Helmy, "A Survey of Mobility Models in Wireless Adhoc Networks," *Wirel. Ad Hoc Sens. Networks*, pp. 1–30, 2004.