# Computational Load Balancing on the Edge in Absence of Cloud and Fog

Saurav Sthapit *, John Thompson *, Neil M. Robertson†, and James R. Hopgood *
* Institute of Digital Communications, School of Engineering, University of Edinburgh
Emails:{s.sthapit, james.hopgood, john.thompson}@ed.ac.uk
†ECIT, Queen's University of Belfast, Email: n.robertson@qub.ac.uk

**Abstract**—Mobile Cloud Computing or Fog computing refers to offloading computationally intensive algorithms from a mobile device to the cloud or an intermediate cloud in order to save resources e.g. time and energy in the mobile device. This paper proposes new solutions for situations when the cloud or fog is not available. First, the sensor network is modelled using a network of queues, then a linear programming technique is used to make scheduling decisions. Various centralised and distributed algorithms are then proposed, which improves overall system performance. Extensive simulations show slightly higher energy usage in comparision to the baseline non-offloading case, however, job completion rate is significantly improved, the efficiency score metric show the extra energy usage is justified. The algorithms have been simulated in various environments including high and low bandwidth, partial connectivity, and different rate of information exchanges to study the pros and cons of the proposed algorithms.

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

## 1 Introduction

The use of *commercial off-the-shelf* (COTS) smart devices in defence and surveillance applications is an extremely useful prospect. Imagine a swarm of COTS devices gathering visual intelligence on a missing person or an armed terrorist (See Fig. 1) using *person re-identification* (PRID) [1] and face identifications [2] algorithms. However, reporting raw data back to a base station can be prohibitive in terms of both time and energy. Furthermore, it may open up the base to external attacks. So some pre-processing must be undertaken on the device itself; for example only reporting to the base once the individual is recognised. For that, the devices must be able to run 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 (Fig.2) such as background subtraction and person detection [3]. The devices may have different computing and energy resources. 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 are outsourced to the cloud, may not be feasible depending on the communication channel to the cloud [4], [5]. Recently, Fog or Edge computing has been introduced whereby mobile devices offload to the nearby servers (preferably at base stations) instead of the cloud [6]. However, Fog computing could be unavailable just like the cloud, for example in underground or underwater scenarios.

In this paper, new algorithms are proposed to balance the computational load among the network of smart cameras for soft real-time applications. For rest of this paper, a network of smartphones running PRID algorithms is considered as the exemplar problem. However, the algorithms can be generalised to other problems such as multistatic radar

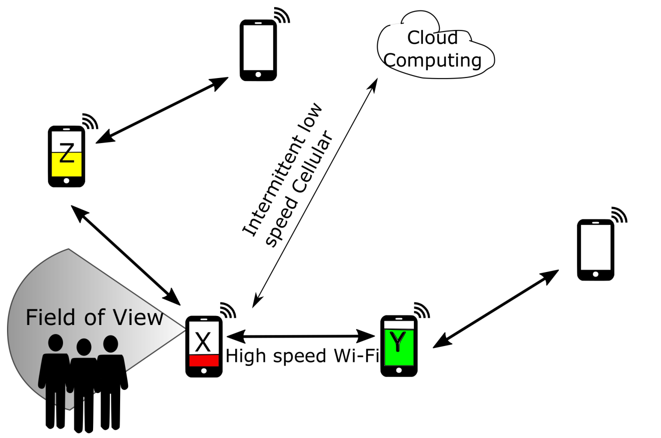


Fig. 1: Pedestrian identification scenario: device X inundated with targets while device Y is idle

or sonar, distributed audio processing etc. The following assumptions are made in this paper:

1) In a network of cameras, targets are spatially and temporally distributed. That means, more targets may appear in some camera FOV's than others and at different times.
2) While targets do not appear in a camera's FOV, its resources (Central Processing Unit (CPU), Graphical Processing Units (GPU)) are not fully utilised. Therefore, in theory, it should be able to help its busy neighbours to cope with the demand.
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 argument about helping neighbours is valid especially if the devices are battery powered. For example, solar powered devices would be recharged every day or a drone swarm would be recharged after $20-30$ minute of flight time. It does not benefit to have energy left when recharging is available. In case of uneven load, by helping neighbours, the network lifetime (the time when the first node in the network runs out of battery) can be extended.

The problem tackled in this paper is twofold. First, a scheduling decision algorithm for *offloadable* jobs (see section 2) among the nodes is created. Second, a determination of the required Node State Information (NSI) (described in Section 3) that needs to be shared in order to make the scheduling decision as well as the update frequency. Queuing theory (see Section 3) is used to model the nodes processing. It abstracts the scheduling algorithms of the underlying hardware so the system may consist of CPU nodes or dedicated accelerators such as Graphical Processing Units and Field Programmable Gate Arrays. Also, by working with job rate rather than individual jobs, the need to take the decision for each and every task is eliminated.

After modelling, the scheduling decision is posed as a minimum cost problem. Based on where the solver is executed and how data is shared, four novel algorithms are presented and their performances are compared with the non-offloading case. The core algorithms were first presented in [7]; this work substantially extends these ideas with further experimentation with real data as well as experiments with more dynamic scenarios such as partial connectivity, and the effect of communication bandwidth. In summary, the main contributions of this paper are:

- Proposed novel algorithms for on-line workload balancing for real-time applications in distributed systems.
- Proposed an Offloading Cost function that incorporates NSI such as battery level, bandwidth and CPU availability.
- Proposed proactive and reactive strategies for sharing NSI among sensor nodes.
- Demonstrate that the proposed algorithms improve the performance of the overall network of battery-powered sensor system compared to the Non-Offloading (NO) system on simulated data as well as a real dataset.

The next section presents the background and related works in MCC. In Section 3, the node network is modelled using a network of queues and the problem is formulated along with the NSI. In Section 4, the algorithms are proposed. Section 5 introduces the simulator developed for testing the algorithms. In Section 6, the experiments and results are presented. Finally, discussions and conclusions of the results and findings are presented in Section 7.

## 2 Related Works

In this section, a brief introduction to Mobile Cloud Computing (MCC) and related works is provided. The main objective is to present the existing works in relation to offloading to neighbouring nodes and the additional challenges. Traditional MCC refers to the offloading of computationally intensive algorithms from a mobile device to the cloud in order to save processing time and energy on the mobile device. Recent literature reports significant time and energy resource saving by offloading to the *cloud* [8], [9]. For a comprehensive list of MCC algorithms, interested readers should refer to the recent surveys [10]–[12]. However, for *offloading to cloud* to have a positive impact, the environment has to be suitable as well. It was discussed in the paper [4] that it may be better to offload to neighbours depending on the bandwidth. These factors apply to computation offloading to neighbouring devices also and is described below.

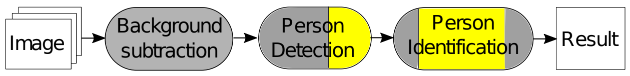


Fig. 2: Typical pedestrian identification flowchart showing *offloadable* and *non-offloadable* algorithm parts in light and dark gray colours respectively.

### 2.1 Characterising Offloadable Algorithms

The benefit of offloading a particular algorithm depends on the speedup that can be achieved as well as the bandwidth available to the cloud [11]. The jobs arriving at the node can be *offloadable* or *non-offloadable* depending on whether the *offloader* can save *time* or *energy* by offloading the job to others. Also, some algorithms are *non-offloadable* because they are inseparable from the device. For example, Operating System (OS) hardware related jobs cannot be offloaded.

Generally, MCC implementations use static and dynamic application partitioning of algorithms based on profiling [13]. For this work the jobs are classified as *offloadable* or *non-offloadable* by design. For example, a typical *person re-identification* software chain is shown in Fig. 2. In this chain, only *person re-identification* is considered as the *offloadable* as its time complexity far outweighs others in the chain and data it requires is minimal (person's image) [3]. However, instead of a binary classification of each algorithm as *offloadable* and *non-offloadable*, each algorithm could be granularised into many sections. Each of the section can be *offloadable* or *non-offloadable*. For example, in Fig. 2 it can be assumed that light gray sections can be offloaded and dark gray sections do not benefit from offloading.

### 2.2 Communication Channel

The availability and quality of a communication channel have a huge impact on successful offloading. Cuervo [8] points out significant energy usage when the Round Trip Time (RTT) increases between the *offloader* and *onloader*. In that sense, offloading to the neighbouring nodes is better than the cloud as the RTT can be expected to be in the range of 10 ms in a typical case compared to $100ms$ for the cloud. Wu et. al [14] also used a queuing theory approach for MCC, however, their focus was on offloading to the cloud and availability of communication channels. Zhang et al [15] used Markov Decision Process (MDP) to tackle the intermittent channel availability. Similarly, many game theoretic approaches also exist whereby nodes compete against each other while using the shared communication channel to avoid interference [16], [17]. In this approach, communication is between neighbouring nodes connected by WiFi or

Bluetooth etc. As the WiFi and Bluetooth coverage is limited compared to cellular network coverage, interference may be limited as well.

### 2.3 Offloading Candidates

The majority of work reported in the literature considers *cloud* and *fog* as the only offloading candidate with the assumption that the *cloud* has unlimited computational resources. As the cloud is mains powered, it is not limited in terms of energy. As such, the decision is mainly limited to "*given current channel availability should you offload or not?*" However, offloading to computationally similar devices needs to answer additional questions such as *"which neighbour is best suited?"* and *"is someone going to offload to me as well?"* Still, as the development of embedded devices continues, researchers are keen to exploit it. For example, Lin et al. [18] considered offloading to *coprocessors*. Authors of [19]–[21] considered offloading to cloudlets along with the *cloud*. Recently, [22], [23] also considered smartphones as offloading candidates. The main objective of [22] is to divide a computationally expensive work into pieces and offload to neighbours. Similar to this work, the cost function comprises computing cost and communication cost and uses an optimisation algorithm to solve the problem. However, the differences are significant, for example, their main aim is to reduce the higher cost incurred due to neighbours moving away from the offloader (uncertainty of connection time), whereas, for this work, the main objective is to balance the computational load among the nodes (uncertainty of target distribution). Their approach is based on the point of view of a single user. Each user care about their own goal only to save their resources so they are termed as "selfish". They do not propose how or when resource discovery is accomplished. This work considers various centralised and distributed approaches with various data exchange policies which show how they can affect the performance.

### 2.4 Summary

The main advantages and disadvantages of offloading to cloud vs offloading to neighbouring nodes are summarised in Table 1. In case of higher bandwidth between neighbouring nodes is based on the availability of WiFi among neighbouring devices whereas only low-speed cellular is available to the cloud. Clearly, only in some cases, neighbouring nodes have benefits over the cloud. However, as it was stated earlier in this paper, the cloud may be unavailable due to several reasons such as natural disasters, terrorist attack, and remote environments etc. In the next section, the sensor nodes are modelled and problem is formulated so that the neighbouring nodes can be considered as offloading candidates and various solutions are proposed.

TABLE 1: Relative comparison between offloading to cloud or fog and offloading to neighbouring nodes. Superior choice is highlighted in bold.

| | **Cloud, Fog** | **Neighbouring nodes** |
|---|---|---|
| Computational capability | **Almost Unlimited** | Limited |
| Energy Limited | **No** | Yes |
| Configuration | **Static** | Dynamic |
| Round Trip Time (RTT) | Long (100ms) | **Short (10ms)** |
| Bandwidth | Lower (1 Mbps) | **Higher ( 54Mbps)** |
| Count | Low (Single) | **Multiple** |

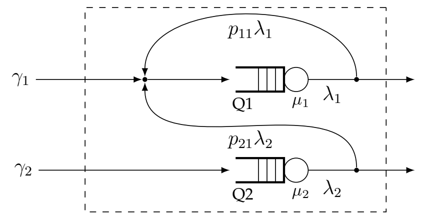


Fig. 3: A network of two Queues. Total incoming target rate at Q1 ($\lambda_1$) is the sum of external target rate ($\gamma_1$) and targets rates emanating from the queues heading to Q1. Under stable condition, outgoing rate is equal to the incoming rate.

## 3 System Model

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 that denotes cost per unit flow on that arc. A link can be single hop or multi-hop. Before going into the detail modelling of sensor nodes, a brief description of a network of queues is presented in the next section.

### 3.1 Network of Queues

Sometimes it is easier to model a system with multiple nodes, with each node having a room for queuing and each having a service centre [24]. Such network of queues is defined as an open network if there are external jobs coming into the system and can be modelled using the Open Jackson network [24]. For example, Fig. 3 shows an open network with two $M/M/1$ queues Q1 and Q2 with external target rates $\gamma_1$ and $\gamma_2$ respectively. The arrival rate for a queue $i \in \{1, ..., n\}$ in such network is given by Eqn. (1).

$$\lambda_i = \gamma_i + \sum_{j=1}^{n} p_{ji}\lambda_j \tag{1}$$

where $\gamma_i$ is the rate of arrival of external targets at queue $i$, $\lambda_j$ is the arrival rate at queue $j$,$p_{ji}$ is the probability a job moves from queue $j$ to $i$. Vilaplana et al. [25] used the Open Jackson to model the cloud architecture and estimate their performance. Based on this formulation, the incoming and outgoing job rates of all the sensors in the system are modelled in the next section.

### 3.2 Node

Each node $i$ is a smart camera with limited computational capability. As an exemplar, this work considers each node to be a COTS smartphone with a CPU, WiFi, cellular link and a camera – see Fig. 4. Two types of queues, $M/M/1$ and $M/M/1/K$ are used to model the behaviour of these components. The $M/M/1$ has First Come First Service (FCFS) scheduling discipline, an arrival process that is Poisson distributed, and a service time that is exponentially

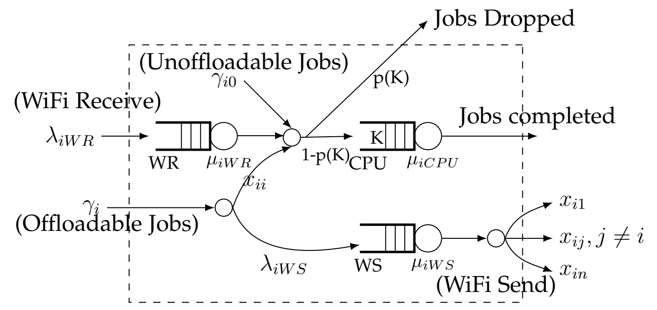


Fig. 4: A sensor node modelled as network of queues. CPU, WR, WS represent CPU, WiFi Receiver and WiFi Sender queues respectively.

distributed [24]. The communication part is modelled using two $M/M/1$ queues (sender and receiver side). The CPU is modelled using $M/M/1/K$ type queue, which is same as the $M/M/1$ except for the finite buffer of size $K$. There can be a maximum of $K$ jobs at any time in the CPU (waiting jobs + jobs being serviced). Size of the buffer for the CPU is chosen so that the system is stable at all times. In this work, $K$ is chosen as

$$K = \left\lceil \frac{T_{\text{threshold}}}{T_{\text{min}}} \right\rceil \tag{2}$$

where $T_{\text{threshold}}, T_{\text{min}}$ are allocated time window and the minimum processing time for each jobs. In the literature, jobs arriving after the buffer is full can be dropped or block preceding queues [26], [27]. In this work, regarding the real-time execution of the application, if a job arrives when the existing number of jobs in the queue equals $K$, the job is not processed and dropped. The probability of dropping a job is $p(K)$ and the job arrival rate for the queue is restricted to the $1 - p(K)$. However, when the buffer is restricted, the Open Jackson network is not valid as it assumes infinite buffer queues. So, for the modelling purpose, they are approximated as $M/M/1$ queues based on the decomposition method proposed by Takahashi et al. [28]. In this method, the arrival rate and service times of the queues are updated based on the dropping probability and the blocking times respectively.

Each node $i$ may be defined as a tuple $\{\gamma_i, \gamma_{i0}, \mu_i, \mu_{iWR}, \mu_{iWS}\}$ where $\gamma_i$ is the rate of *offloadable jobs*, $\gamma_{i0}$ is the rate of *non-offloadable jobs*, $\mu_{iCPU}$ is the service rate of CPU and $\mu_{iWR}, \mu_{iWS}$ are the WiFi transmission rates. This node information is defined as the Node State Information (NSI). Each individual target that passes through a camera 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*. The notations and their definitions are listed in Table 2.

## 3.3 Centralised Problem Formulation

The scheduling decision problem is defined as a minimum cost flow problem to find the optimal policy $X$ such that all the jobs get scheduled among the available nodes with minimum energy and time costs and with constraints that all the jobs get scheduled, without compromising the stability of the queues. The optimal policy $X$ is given by

$$X = \underset{x}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{3a}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = \gamma_i, \qquad \forall i \in N \tag{3b}$$

$$\sum_{j=1}^{n} x_{ji} + \gamma_{i0} \preceq \mu_{iCPU}, \qquad \forall i \in N \tag{3c}$$

$$x_{ij} \geq 0 \tag{3d}$$

The decision variable $x_{ij}$ represent the probability of job flow on an communication link $(i, j) \in A$ and $x_{ii}$ is the job rate that is executed locally. $c_{ij}$ represents the general cost of scheduling a job from node $i$ to $j$ which is described in detail later in Section 3.5. The solution of Eqn. (3) can be written as a decision matrix shown below:

$$X = \begin{bmatrix} x_{11} & . & x_{1i} & . & x_{1n} \\ . & . & . & . & . \\ x_{i1} & . & x_{ii} & . & .x_{in} \\ . & . & . & . & . \\ x_{n1} & . & x_{ni} & . & x_{nn} \end{bmatrix} \tag{4}$$

Each row of $X$ represents the policy for each node. We defined it as a *decision vector*(dv). The dv$_i$ tells node $i$ how it should process the incoming targets. Also, $i^{th}$ column of the matrix indicates the policy of other nodes towards the $i^{th}$ node. The rate stability of a queue can be guaranteed 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, an alternative node has to sought. The equality constraint in (3b) makes sure that all the jobs are assigned to a processing node whereas the inequality constraint in (3c) makes sure that the jobs can be processed by the corresponding nodes they are assigned to. This formulation uses NSI from all the nodes ($N$) and makes decision for all the nodes simultaneously. Eqn. (3) can be solved using efficient linear programming techniques [29].

TABLE 2: List of Notations.

| Notation | Definition |
|---|---|
| $N$ | Set of sensor nodes {1,...,n} |
| $A$ | Set of directed arcs between nodes. |
| $\gamma_i$ | Incoming external Offloadable jobs rate of $i^{th}$ node |
| $\gamma_{i0}$ | Incoming external Unoffloadable jobs rate of $i^{th}$ node |
| $\lambda_{iCPU}$ | Total incoming job rate for $i^{th}$ CPU |
| $\mu_{iCPU}$ | Job service rate of CPU of $i^{th}$ node |
| $\lambda_{iWS}$ | Total incoming job rate of WiFi send queue for $i^{th}$ node |
| $\mu_{iWS}$ | WiFi transmission rate of $i^{th}$ node |
| $\lambda_{iWR}$ | Total incoming job rate of WiFi receive queue for $i^{th}$ node |
| $\mu_{iWR}$ | WiFi receive rate of $i^{th}$ node |
| $f$ | average retransmission times |
| $BW_{ij}$ | Expected bandwidth between node $i$ and $j$ |
| $B_i$ | Remaining battery in node $i$ |
| $L_i$ | Number of CPU Jobs in Node $i$ |
| $T_i$ | Average processing time for each CPU Jobs |
| $L_{iWS}$ | Jobs in WiFi send queue of node $i$ |
| $L_{iWR}$ | Jobs in WiFi receive queue of node $i$ |
| $T_{iWS}$ | Expected time to process one WiFi job $i, j$ |
| $\omega_1, \omega_2, \omega_3$ | Weighting factor set to $0.6, 0.3, 0.1$ respectively |

## 3.4 Distributed Problem Formulation

In a large network, collecting NSI from all the nodes may not be advised for several reasons. For example, collecting NSI information and sending $\mathrm{dv}_i$ may have significant impact as the bandwidth decreases and the frequency of information exchange increases. Also, nodes that cannot be reached due to lack of communication links, can neither offer help nor ask for help. So, the centralised problem is simplified by *primal decomposition* [29] whereby each node calculates its own dv. The distributed formulation can then be defined for each node $i \in N$ as given by Eqn. (5).

$$\mathrm{dv}_i = \underset{x}{\operatorname{argmin}} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{5a}$$

subject to,

$$\sum_{j=1}^{n} x_{ij} = \gamma_i \tag{5b}$$

$$\sum_{i=1}^{n} x_{ji} + \gamma_{i0} \preceq \mu_{iCPU} \tag{5c}$$

$$x_{ij} \geq 0 \tag{5d}$$

This is similar to the *Gauss-Siedel* like method used by Meskar [30] for MCC. The algorithm basically communicates with its immediate neighbours to ask for help and makes the decision. The approach is not selfish as it still considers neighbours' resources rather than offloading everything. It is different from the centralised problem in Eqn. (3) where each node $i$ only tries to minimise the cost of its own objective function on the basis of information available on its neighbours. Eqn. (5) can also be solved using linear programming techniques [29].

## 3.5 Cost Function

Once all the arriving jobs can be scheduled such that the queues are all rate stable, it should be accomplished with the minimum cost. Here, the cost function $c_{ij}$ used in both central and distributed formulation described by Eqn. (3) and Eqn. (5) is defined. It is composed of energy costs in the communication links, availability of the CPU and the remaining energy. More precisely, the cost of scheduling from node $i$ to $j$ is defined as:

$$c_{ij} = \begin{cases} \omega_1 L_i T_i, & \text{if } i = j \\ \omega_1 L_j T_j + \omega_2 \alpha_{ij} + \omega_3 \frac{1}{B_j}, & \text{if } i \neq j, (i,j) \in A \\ \infty, & \text{if } i \neq j, (i,j) \notin A \end{cases} \tag{6}$$

where, $L_i$ is the number of CPU jobs already in node $i$, $T_i$ is the average processing time of each CPU Jobs,$B_j$ is the remaining energy in node $j$ (Joules) and $\{\omega_k\}_1^3$ are weight factors. The significance of various components in Eqn. (6) can be changed using the weighting factor $\{\omega_k\}_1^3$. Even though the selection of weights could be unique based on application and context, we describe a general methodology on their selection. The weights can be static for a system or dynamically varying depending on context. For example, for the nodes that are mains powered, $\omega_3 = 0$ can be selected universally . This would mean that overall cost of offloading would be cheaper to the nodes that have mains power at the time. Similarly, some nodes may have prior knowledge of incoming target density, which could have been learnt over time. For example, cameras monitoring entry and exit of a station would be busier during office hours. Those nodes can set high value for $\omega_1$ so that other nodes do not offload to them at those times. In the same manner, nodes with less battery power can minimize the load distributed to them without fully isolating themselves from the network by selecting higher value of weights. This would mean that other nodes would only offload to them if there is no one else available to help. Likewise, if the communication bandwidth is expensive or currently required for some other service, the $\omega_2$ could be set high so that the nodes would prefer on-board processing to offloading. The weights are set at $0.6, 0.3$ and $0.1$ for this application. Each component of the cost function is described in the next section.

### 3.5.1 CPU Availability

The number of existing jobs in the CPU queues ($L_i$) is used as the measure of CPU availability in the node. A higher number indicates lower availability for further external jobs and vice versa. This cost is applicable to self-processing in the scheduling decision making as well.

### 3.5.2 Communication Cost

The WiFi communication cost (time as well as energy) depends upon the bandwidth between the nodes and data size. However, the communication channel is not perfect due to various noises and interference. Bandwidth is adjusted depending on these factors using metrics such as Signal to Noise Ratio (SNR), acknowledgement etc. for optimal performance which is to offer high bandwidth at high Packet Delivery Rate (PDR) [31], [32]. Results from [32] show that depending on SNR, the PDR can be different for different data rates. So in order to model their behaviour correctly, this paper accounts for them using a retransmission factor $f$. In the experiments, PDR is randomly sampled between two nodes and uses the mean of the geometric distribution to calculate the average number of transmissions to send the data from one node to another (see Eqn. (7a)):

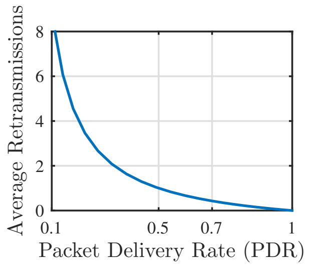
$$f(\mathrm{PDR}) = \mathbb{E}[g(x; \mathrm{PDR})], \text{ where} \tag{7a}$$

$$g(x; \mathrm{PDR}) = \mathrm{PDR}(1 - \mathrm{PDR})^{x-1}, \forall x \in \{0, .., \infty\} \tag{7b}$$

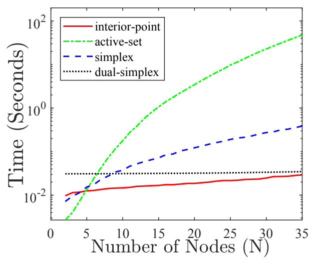
The relationship (see Fig. 5a) shows us that as the PDR degrades, the average number of retransmission rises exponentially. For example, if the PDR is $1, 0.5$ and $0.1$, the average number of times the data has to be transmitted is $1, 2$ and $9$ times, respectively. In Section 4.2, further analysis is performed to see effect of bandwidth, PDR and frequency of data exchange on the communication resources. For the simulations, $0.5$ is considered as the minimum PDR for any valid communication link. Then the communication cost between $i$ and $j$, $\alpha_{ij}$ is defined as

$$\alpha_{ij} = L_{iWS} T_{iWS} + D \times \frac{f+1}{BW_{ij}} + L_{jWR} T_{jWR} \tag{8}$$

where $BW_{ij}$ is the bandwidth between node $i$ and $j$; $D$ is the data size; $f$ is the average retransmission times (see Eqn. (7a)); $\alpha_{ij}$ is the communication cost; $L_{iWS}, L_{jWR}$ are the number of jobs already in the WiFi send and receive



(a)



(b)

Fig. 5: (a) Average no. of retransmissions required due to imperfect channel. (b) Time complexity of various linear solvers.

queues of node $i$ and $j$; $T_{iWS}, T_{jWR}$ are expected WiFi sending and receiving time in $i$ and $j$. Note that $\alpha_{ij}$ can be interpreted as the suitability of node $j$ based on existing communication queues and the channel available.

#### 3.5.3 Energy available

The last element of the cost function is the battery level of the *onloader*. When the battery level at node $j$ is close to full, it does not affect the decision making significantly due to the large value of $B_j$ in Eqn. (6) as the corresponding term is small. However, when the battery is nearly empty, its significance is considerably higher. It makes our decisions "energy aware" i.e. the nodes do not completely drain while trying to help the neighbouring nodes. Detailed models of power drain for the CPU, Image sensor and WiFi communications are described in section 5.2.

### 3.6 Computational Complexity of Optimisation

It would be inefficient if the proposed optimisation algorithms uses a significant amount of CPU resources itself to balance the computational load. Fortunately, the optimisation problem stated in Eqns. (3) and (5) can be solved using efficient linear programming techniques. Using data rate in the problem formulation means that we do not have the integer constraint and the decision can be taken periodically rather than for each and every job as they arrive. Experiments were performed to gauge their *time complexity* for a different number of nodes. 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 an embedded device may be significantly higher but should follow the similar pattern. The results show that the Interior Point is the most efficient for 5 to 35 nodes –see Fig. 5b. Running Optimisation periodically incurs time and energy cost on the node. In this work, we assume the size of network to be around 30, so a fixed cost of optimisation is accounted. This is similar to executing a general algorithmic task running on the node which is described in Section 5.1. Every time a node runs the optimisation algorithm, it's costs are accounted by the simulator. The overall performance of the system is dependent on the frequency of the optimisation algorithm execution. This is described in Section 6.4.

## 4 Algorithms

In section 3.3 and 3.4, the problem of scheduling jobs was formulated as a *centralised* and *distributed* problem. This section describes how those solutions are implemented. Two data sharing mechanisms; *proactive* and *reactive* are also considered. Depending on which solution is used, and how the data is shared amongst the nodes, four algorithms are proposed. All four algorithms are then compared to the Non-Offloading case when offloading is not allowed whatsoever. For this work, a co-operative environment is assumed, such that every node wants to achieve global objectives (i.e. process the most jobs in an allocated time). Also, by "co-operative", it implicates that: if a node sends a job to another node, the other node must execute it (see Eqn. (3) and (5)). However, an assumption is made that the nodes are not selfish and only offloads if required.

### 4.1 Oracle (O)

The target detection rate varies with time so the job rates ($\gamma$) in Eqn.(3) and (5) are non-stationary. The lowest sampling time of the simulator is $10ms$ matching a typical RTT, hence the problems in Eqn.(3) and (5) must be solved periodically. For the *Oracle*, it is assumed that it has access to every sensor Node State Information (NSI) at all times. Since it has no energy limitation, the *Oracle* solves the cost minimization problem in Eqn. (3) every second which is every hundredth sampling step. Once solved, it sends the related policy $\mathrm{dv}_i$ to all nodes simultaneously without using the communication channel. While this continued update of NSI, is not feasible in practice, it provides a benchmark for comparison.

### 4.2 Proactive Centralised (PC)

This is a more realistic version of the *Oracle*. In this method, a node from among the nodes, is nominated as the *server* and all other $(n-1)$ nodes send their NSI to it. Similar to Oracle, the server then solves Eqn. (3) and sends the corresponding policy (dv) back to each node. All other nodes are obliged to follow the decision made by the *server* and computes and offloads based on the policy $\mathrm{dv}_i$ until a new one is broadcast. However, different and in contrast to Oracle, the cost of communication, as well as cost of executing the solver periodically, are taken into account. Section 3.6 show that the cost is fairly constant when the number of nodes are up to 20 sensors, we add this to the CPU queue as well.

An important distinction with the Oracle is that, due to the partial connectivity among the nodes, some of the sensors are not able to communicate to the *server* and vice-versa. Hence they are excluded from the offloading process altogether. In order to minimise this effect and minimize extra drain of the *server*'s energy, a new server is selected in *round-robin* basis. In this paper, every minute a different *server* is chosen which acts as the *server* and so on.

A key question that arises is how often the nodes need to broadcast their NSI and how often can they broadcast it without flooding the communication links. Obviously, the answer depends on many factors such as the communication bandwidth, size of NSI, PDR and number of nodes in the set. If there are $n$ nodes in total, and $n-1$ nodes sending their NSI to the *server* every $t$ seconds, the node with the

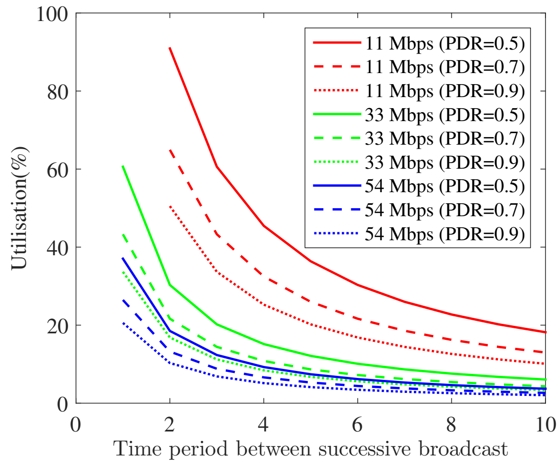


Fig. 6: Queue utilisation of *server* in proactive setting under various network conditions (Lower is better). Data size set at 1 Mb.

highest probability of being busy is the *server*. The arrival rate, worst service rate and the utilisation of the *server*'s receiving queue can be calculated as follows:

$$\text{Arriving rate}, \lambda = \frac{n-1}{t} \tag{9}$$

$$\text{Worst Service rate}, \mu = \frac{\text{Data Rate} \times \text{worst PDR}}{\text{NSI size}} \tag{10}$$

$$\text{Utilization}, \rho = \frac{\lambda}{\mu} = \frac{(n-1) \times \text{ NSI size}}{t \times \text{Data Rate} \times \text{PDR}} \tag{11}$$

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

where, $p[0]$ is the probability that there is no jobs in the queue. Based on the arriving rate and service rate, the utilisation of the WiFi receiver queue of the *server* can be estimated. Low utilisation is desired as it means lower delay and more room for transmission of other data. 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. (7a) estimates the queue utilisation is $\approx 0.03$ and no waiting times for $\approx 97\%$ of the time. Similarly the average delay is around $\approx 0.03$ seconds. Fig. 6 shows waiting times at the receiving node at various intervals and for different speeds. For the data rate of 11 Mbps (red lines in Fig. 6) any PDR and NSI frequency leads to significant usage of communication resources which is not desirable. However, for 33 and 54 Mbps, NSI exchanges can be frequent upto once every five seconds, without significantly using the communication resources.

### 4.3 Proactive Distributed (PD)

Proactive Distributed (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. Time and energy cost of solver is also on each node.
2) Instead of solving central problem in Eqn.(3), each node only solves distributed problem in Eqn.(5).
3) Set $N$ contains immediate rather than neighbours than all the nodes. Even if total nodes is large ($> 100$), $N$ may be limited to tens of nodes. For example, see Fig. 7b, node 1 and 5 are only connected to one another.

### 4.4 Reactive Distributed (RD)

If a few nodes become overloaded 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 [33]. In this method (see Alg. (1)), nodes only communicate when they need to offload. The node seeking offloading 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. (5). To avoid using old information and update neighbour's current situation, a timer $T_{th}$ is set after which the NSI expires and the node has to start again by broadcasting the RFH.

**Algorithm 1** Reactive Distributed algorithm

```
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.(5) for new dv$_i$ and follow it.
        else
            Broadcast RFH again, follow previous dv$_i$.
        end if
    end if
end if
```

## 5 Simulator Setup, Data and Network

In our previous work, a simulator was developed to run offloading algorithms [4]. It is defined here again briefly for completeness. The simulator consists of a three-dimensional space called the platform. Sensors are stationary and placed on the platform base ($z = 0$) randomly during initialisation. One instance of the resulting simulator setup is shown in Fig. 7. Fig. 7a shows sensor placement and Fig. 7b shows how they are connected to each other. The connection links are created based on the sensor positions. Targets spawn in the platform and move around (see Section 5.3.1). When the targets comes to the FOV of a sensor, it gets detected; once detected, the sensor has to identify the target within the allocated time. The major elements of our simulator relate to the algorithmic tasks, the sensor architecture, communication links and the targets.

### 5.1 Algorithmic Tasks

Execution of an algorithm on a modern CPU is a complex process. Apart from the number of Operation (OP)s required

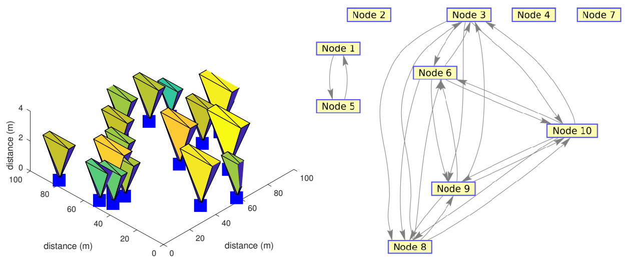


Fig. 7: Simulation setup for one monte-carlo simulation. (a) Ten sensors (blue squares) with uneven FOV placed randomly on the simulation platform of $100m \times 100m$ size . (b) Visualising sensor connectivity based on spatial positioning.

TABLE 3: Execution details for a bodytrack example in PARSEC [34] consisting 4 frames and 4000 particles.

| **Instructions(Billions)** | | | **Synchronization Primitives** | | |
|---|---|---|---|---|---|
| **Total** | **Reads** | **Writes** | **Locks** | **Barriers** | **Conditions** |
| 14.03 | 3.63 | 0.95 | 114,621 | 619 | 2042 |

to execute the algorithm, an execution on a CPU depends upon several factors such as multi-stage pipeline, cache-miss rate and parallelism etc. The Princeton Application Repository for Shared-Memory Computers (PARSEC) [34] benchmark suggests typical applications have billions of instructions to execute with an equally large number of read and write operations. For example, Table 3 details the execution details including synchronization primitives for a body tracking application from the PARSEC benchmark [34]. However, to keep the simulator simple, an algorithmic task is characterised just by its number of OPs, input and output data size. For example, a person detection algorithm takes an image of size $M \times N$ as the input, requires approximately $C$ OPs per image and outputs the number of persons in the image. Assuming one OP per clock cycle, the execution time on the device can be estimated using the clock frequency.

$$T_{\text{exec}} \propto \frac{C}{\text{Clock Frequency}} \tag{12}$$

## 5.2 Component Based Sensors

In order to realistically emulate its behaviour, a sensor is divided into its components such as the CPU and cellular radio. The utilisation based model by Jung et al. is implemented to calculate the energy consumption [33] and our parameters are based on a Google Nexus I phone which was a Device Under Test (DUT) in [33]. If desired, the simulator can be easily calibrated for a different DUTs.

### *5.2.1 Application Processor (AP)*

The CPU power consumption is made up of two parts, idle power and the running power, as follows:

$$P^{\text{cpu}} = \beta^{\text{cpu}}_{\text{freq}} \times u + \beta^{\text{cpu}}_{\text{idle}}, \tag{13}$$

where $u$ is the utilisation and $\beta^{\text{cpu}}_{\text{freq}}$ and $\beta^{\text{cpu}}_{\text{idle}}$ are the CPU parameters, listed in Table 4 for the DUT [33]. The utilisation

TABLE 4: CPU Parameters for the DUT (Google Nexus I) based on Jung et al. [33].

| Freq. | 245.0 | 384.0 | 460.8 | 499.2 | 576.0 | 614.4 | 652.8 | 691.2 | 768.0 | 806.4 | 844.8 | 998.4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\beta^{\text{cpu}}_{\text{freq}}$ | 201.0 | 257.2 | 286.0 | 303.7 | 332.7 | 356.3 | 378.4 | 400.3 | 443.4 | 470.7 | 493.1 | 559.5 |
| $\beta^{\text{cpu}}_{\text{idle}}$ | 35.1 | 39.5 | 35.2 | 36.5 | 39.5 | 38.5 | 36.7 | 39.6 | 40.2 | 38.4 | 43.5 | 45.6 |

is calculated as the ratio of the CPU time used vs. the time available per frame. However, the CPU is also used by the OS and other running applications. Dargie [35] used normal and exponential distributions to simulate workload. Also a random variable, $r$ sampled from a Gaussian distribution is used to simulate these other activities. By adjusting the mean of $r$, busy and idle sensors can be simulated. The total utilisation is calculated as:

$$u = \frac{\sum_{i=1}^{N} T_{\text{exec}\,i}}{T_{\text{Frame}}} + r \tag{14}$$

where $N$ is the number of algorithms to be processed, $T_{\text{exec}_i}$ is the execution time for $i^{\text{th}}$ algorithm for execution times for all algorithms) and $T_{\text{Frame}} = \frac{1}{\text{FPS}}$ is the time available for each frame. In the situation where $T_{\text{exec}_i} > T_{\text{Frame}}$ which is very likely in the case of algorithms for person re-identification; the CPU is run up to $100\%$ load and run the remainder of the algorithm in the next frame and so on.

### *5.2.2 Image Sensor*

The image sensor consumes significant energy in a mobile device when used continuously. According to Likamwa et al. [36], when using the image sensor continuously, the energy consumption per frame of the image sensor can be modelled as:

$$E_{\text{camera}} = P_{\text{idle}} \times (T_{\text{frame}} - T_{\text{active}}) + P_{\text{active}} \times T_{\text{active}} \tag{15}$$

where, $T_{\text{frame}} = 1/\text{FPS}$ is time allocated for each frame, $T_{\text{active}} = \text{Number of Pixels/Camera Clock Frequency}$ is the time taken by the sensor to gather the pixel data, and $P_{\text{idle}}, P_{\text{active}}$ are the idle and the active power consumption of the image sensor respectively. Based on Eqn. (15), power consumption of the image sensor depends on image resolution and the acquisition rate. The parameters used for the simulation are listed in Table 5.

### *5.2.3 Wi-Fi*

The Wi-Fi model calculates the time and energy of the Wi-Fi component in the connected mode. There are two modes depending upon the packet rate.

$$p^{\text{wifi}} = \begin{cases} \beta_{\text{LT}} \times p + \beta_{\text{LT base}} & \text{if } p \leq P_{Th} \\ \beta_{\text{HT}} \times p + \beta_{\text{HT base}} & \text{if } p > P_{Th} \end{cases} \tag{16}$$

where $p$ is the packet rate, $\beta_{\text{LT}}, \beta_{\text{HT}}, \beta_{\text{LT base}}, \beta_{\text{HT base}}$ and $P_{Th}$ are the parameters of the DUT based on [33] (see Table 5). As per [33], if the number of packets per second exceeds the threshold of 20 then Wi-Fi is in the high power state, else in the low power state. Unlike the cellular system, the power consumption is directly proportional to the data rate.

TABLE 5: Image Sensor and WiFi Parameters

| Image Sensor | | WiFi | |
|---|---|---|---|
| *Parameter* | *Value* | *Parameter* | *Value* |
| $P_{\text{idle}}$ | 225.4 Joules | $\beta_{\text{LT base}}$ | 238.7 |
| $P_{\text{active}}$ | 338.8 Joules | $\beta_{\text{HT base}}$ | 247.0 |
| Image Resolution | $800 \times 600$ | $\beta_{\text{LT}}$ | 1.2 |
| Camera Clk Frequency | 32 MHz | $\beta_{\text{HT}}$ | 0.8 |
| | | $P_{Th}$ | 20 pkts/sec |

## 5.3 Target Data

The proposed centralised and distributed algorithms defined in Section 4 along with the Non-Offloading (NO) case, are tested on two different datasets. The first is a simulated dataset and uses a widely used mobility model called Random Waypoint Model (RWP), and the second uses real data from a computer vision dataset. They are briefly described below.

### 5.3.1 *Simulated Data: Random Waypoint Model*

In the Random Waypoint Model [37], targets spawn at random locations in the platform and moves around the platform in a straight line. 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 a constant velocity; the process repeats until it dies (i.e. target moves out of the platform). In order to have different job-rate among the nodes, the size of FOV is also randomly selected (see Fig. 7a. The target spawning rate is higher than dying rate, so target rate generally increases over time across all nodes – see Fig. 10a.

### 5.3.2 *Real Data: SAIVT Dataset*

A multi-camera scenario described in SAIVT Multi-Camera Surveillance Database [38] is chosen to test the algorithms on a real dataset. This dataset consists of eight cameras and contains movements of more than 150 people in a cafeteria. The target tracks for the simulator were extracted from the Extensible Markup Language (XML) files provided with the dataset. According to the dataset [38], the acquisition rate was 25 FPS. A brief study of their target distribution revealed there were far too many targets in the short span of time and majority of the targets appeared in the first half of the dataset. So, the FPS was relaxed to 10 and the data was split along the timescale to 16 sensors. The resulting target distribution looked like shown in Fig.8. The majority of targets are detected by Cameras 1, 7 and 15.

## 6 Experimental Results

In this work, 100 Monte-Carlo simulations were executed for $720,000$ simulation steps which is equivalent to 12 minutes of simulated time, on two sets of target data described in section (5.3).

## 6.1 Calculation of RTT

In order to establish that our assumptions about the communication network is valid, some experiments were carried out on Network Simulator 3 (NS-3) [39] for Wi-Fi, and real Wi-Fi and cellular networks. The main objective of this experiment was to see if the delay would be non-negligible

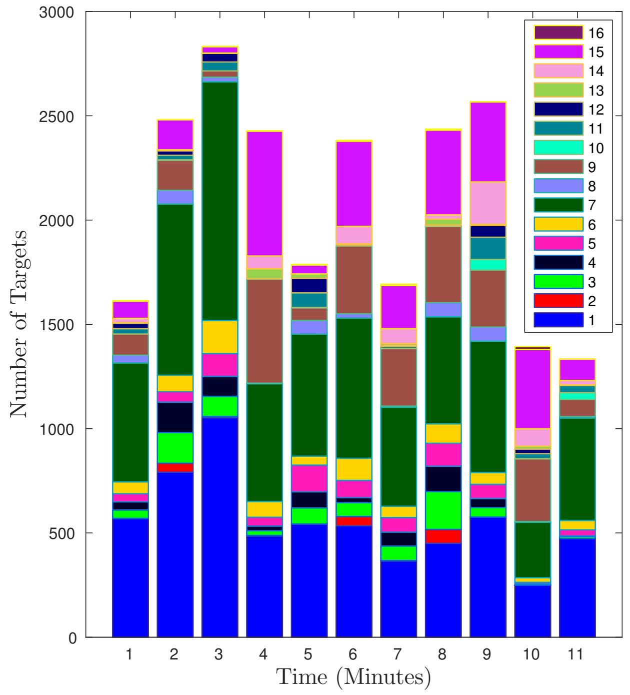


Fig. 8: Heterogeneous loading of cameras in a multi-camera scenario. Each shaded band represents target load on each camera. For example, the bottom and the top bands represent target arrivals in camera index 1 and 16 respectively. Majority of targets appear in camera indexed $1, 7$ and $15$. (Best viewed in colour)

TABLE 6: NS-3 simulation parameters.

| Parameter | Value |
|---|---|
| Network Structure | 1 AP + 2 Stations |
| Network Protocol | 802.11n |
| Modulation | 64 QAM |
| Data rate | 54 Mbps |
| Data rate (ACK frame) | 6 Mbps (MCS0, constant) |
| Error rate model | YANS error rate model [40] |
| Mobility Model | Constant position |
| | Reno-TCP |
| Delay type | Constant propagation delay model |
| Channel Loss model | Friis propagation loss model |
| Transmission power | 10 dBm |
| Distance between stations | 2-5 metres |
| Network traffic | Single queue and full buffer |
| Typical Data Size | $240 \times 120 \times 3 \times 8$ bits $\approx$ 85 KBytes |
| Average Latency | 13.1 Milli Seconds |

as multiple station will be transmitting at the same time. For the NS-3 simulation, we considered three nodes. One acting as the AP and two acting as stations. The two stations simultaneously transmitted data to the AP so it represents the worst case scenario. Further simulation parameters are displayed in Table 6 Fig. 9 shows the distribution of the latency for the scenario of one AP and two stations when the stations are transmitting continuously (worst case scenario). The average delay based on data size of 85 KiloBytes was about 13 Milli-seconds which is well under our threshold of one second. In offloading applications, the stations will not be transmitting at all times, so the probability of collision would be reduced. However, there may be more nodes that may transmit at the simultaneously.

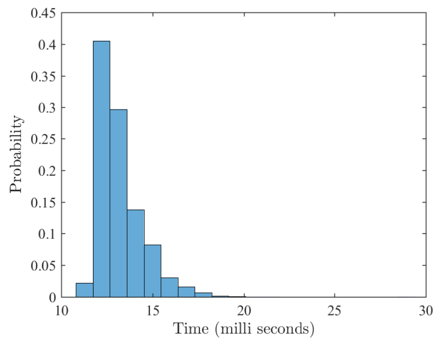


Fig. 9: Latency for NS-3 simulation. The average RTT was 13.1 Milli seconds.

TABLE 7: Simulation Parameters.

| Dataset | Bandwidth (Mbps) | NSI Frequency (sec) | Network Size | Range (Metre) |
|---|---|---|---|---|
| RWP | 1, 11, 54 | 5, 10, 20 | 10 | 30, 60, 90 |
| SAIVT | 1, 11, 54 | 5, 10, 20 | 16 | 30, 60, 90 |

The total energy consumption for each sensor was estimated by summing power consumption of each component based on energy values from Eqn. (13), (15) and (16) in section (5.2). For each run, the simulator was initialised as per Algorithm (2). Each simulation was repeated for the various parameters to see if there is any effect on algorithm performance (see Table 7).

**Algorithm 2** Simulator initialisation.

```
Generate $n$ sensors randomly on the platform.
Create communication links between sensors that are within the communication range.
For each link, randomly generate Packet Delivery Rate
Use shortest path algorithm to calculate cost per bit between nodes. The cost can range between 0 (ie same node) to $\infty$ (i.e. no communication link).
```

## 6.2 Results for the Standard Configuration

Fig. 10a shows the average target detected across all the nodes and across all the trials, normalised by the total capacity of the system for the RWP dataset. It remains same for all the different simulator parameters specified in Table 4. Targets that cannot be processed within the allocated time (30 and 20 for RWP and SAIVT respectively) is considered as dropped targets. At around 10 minutes, the target rate exceeds the computational capacity of the system so even in an ideal case, targets would be dropped. Fig. 10b shows the results for the standard configuration of 11 Mbps, communication range of 60m and NSI exchange every 5 seconds. In the baseline NO case, about 30% of all targets are dropped. The RD does slightly better than the NO and drops only about 25%. The PD however, performs quite well and drops approximately 40% less targets. The performance of centralised algorithms though is at a different level. The PC and the O drops only about 5% and 3% of the targets. Another noticeable fact is that the centralised algorithms dropped only a few targets up to 8 minutes, this is when more targets arrive than the system can process. The results will be further analysed in Sections 6.3 to 6.5

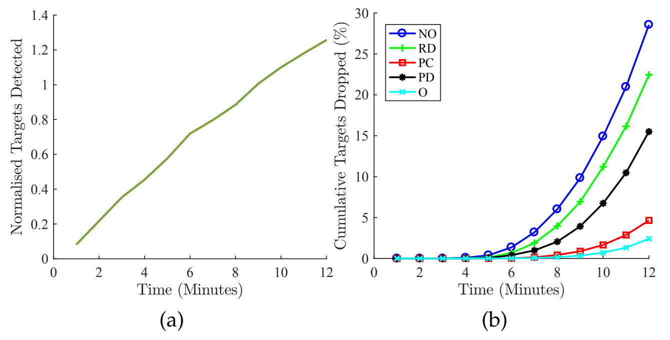
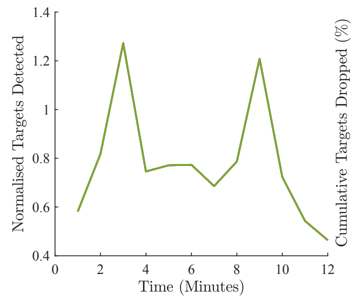


Fig. 10: Simulation results for RWP target data with Bandwidth 11 Mbps NSI exchange frequency of 5 seconds and range of communication limited to 60 meters. (a) Normalised Target arrival rate per nodes over simulation time (b) Targets dropped over Arrival Rate. NO dropped the most (30% of all targets). Centralised algorithms performed best with at least 80% reduction in dropped targets and distributed algorithms perform in between.

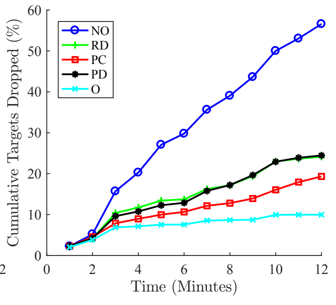
Fig. 11a shows the target arrival rate during the simulation time for SAIVT. Similar to RWP case, it remains constant for different simulation parameters. Unlike RWP, the SAIVT has two peaks during the simulation when the target rate is higher than the maximum processing capability of the system. In this case, the NO algorithm dropped almost 60% of all targets which is very poor. All the proposed algorithms performed significantly better than that. Both distributed algorithms (RD and PD) produced very similar results and in both case the targets dropped were recorded to be around 22% which is less than half of the baseline case. The Oracle performed best followed by the PC solution. They dropped approximately 10 and 20% of the targets respectively. Also, the Oracle method did not drop any significant targets after approximately three minutes which is the first peak in load shown in Fig. 11a. In the remaining section, the performance is analysed with respect to the energy consumed as well as effect of environment and parameter selections.

## 6.3 Process Score and Efficiency Score

The *process score* is defined as the percentage of jobs successfully executed in the allocated times. The *efficiency score (ES)* as the ratio of *Successful Identification* to the energy consumed [4]. Similar metric (mAP/Energy) has also been used by Mao et.al [41] for measuring the performance of their object detection algorithm on embedded platform where mAP is the mean Average Precision. In simple terms, ES is a measure of work accomplished per joule and shows if the extra energy cost is justified (especially for a battery powered device). The overall result is summarised in Table 8 and Fig. 12 for the standard configuration. For Fig. 12, the objective of the proposed algorithm is to be at the top left corner which means the system uses less energy but provides better performance. This is not always possible and some extra energy has to be used to gain performance. The ES metric gives an insight if the extra energy consumed



(a)



(b)

Fig. 11: Simulation results for SAIVT target data with Bandwidth 11 mbps NSI exchange frequency of 5 seconds and range of communication limited to 60 meters. (a) Normalised Target arrival rate per nodes over simulation time. (b) Cumulative targets dropped over time. Proposed algorithms perform significantly better than the NO case. Distributed algorithms dropped less than half of the baseline and the Oracle dropped only about sixth.

is justified and can help in selecting the right algorithm. This can be explained using an example, in Fig. 12a, PD performs slightly better than RD but also uses slightly more energy. Between those two, which one should be preferred? Those two algorithms have ES of 1.07 and 1.12 respectively which suggest that the system achieves better performance per joule using the PD than RD. So PD should be chosen over RD. However, in case of PC and PD, PC is superior as it has a higher ES score. This can be seen in Fig. 12 as well.

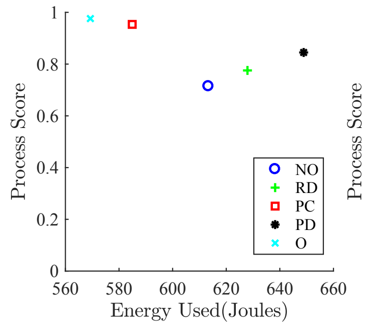
In both datasets, Oracle performs better than the PC, which can be explained by two reasons. First, the Oracle takes decisions every second as opposed to every five seconds in PC. Second, when choosing the *nominated server* in PC on a round-robin basis, due to the partial connectivity, not all the nodes can communicate with the server which results in slightly degraded performance (see Section 6.5). However, PC is still superior than the distributed algorithms. Regarding energy consumption, in the RWP case, the centralised algorithms actually consumed less energy than the NO case. It is because when not offloading some of the sensors were utilised heavily and consumed a lot of energy whereas others were idle which still consumed some energy. By offloading, the load was more balanced and overall the system consumed less energy.

## 6.4 Effect of Bandwidth and NSI Frequency

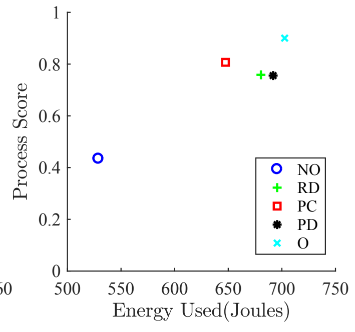
In the RWP simulation, the bandwidth had minimal effect on the performance (i.e. no change in targets dropped overall due to change in bandwidth) – see Fig. 13a. This may be due to the lower amount of data exchanges rather than the bandwidth having no effect at all. This is evident in the real SAIVT dataset case, where the number of targets were significantly higher ( see Fig. 13b). All three algorithms, RD, PC and PD benefited from higher bandwidth but the significance was higher in the case of distributed algorithms. Also, increasing the bandwidth from 11 Mbps to 54 Mbps had minimal effect on the performance but slightly increased energy usage. This can be explained using Eqn. (16), the higher bandwidth led to higher packet rate increasing the radio power slightly. As the data was transmitted periodically, the WiFi radio could not go into the sleep state. Hence the slight increase in energy usage.

TABLE 8: Simulation Results (Averaged over 100 runs) for Bandwidth 11 mbps NSI exchange frequency of 5 seconds and range of communication limited to 60 meters.

| Data | Algo-rithm | Arrival Rate (/min) | Service Rate (/min) | Process Score | Energy Used (Joules) | Efficiency Score (Ident/100J) |
|---|---|---|---|---|---|---|
| RWP | NO | 8.6 | 6.16 | 0.71 | 613 | 1.0047 |
| | RD | 8.6 | 6.69 | 0.78 | 628 | 1.0653 |
| | PD | 8.6 | 7.29 | 0.85 | 649 | 1.1232 |
| | PC | 8.6 | 8.22 | 0.95 | 585 | 1.4061 |
| | O | 8.6 | 8.42 | 0.98 | 569 | 1.4786 |
| SAIVT | NO | 9.37 | 4.10 | 0.43 | 529 | 0.7696 |
| | RD | 9.37 | 7.11 | 0.76 | 680 | 1.0448 |
| | PD | 9.37 | 7.08 | 0.76 | 692 | 1.0237 |
| | PC | 9.37 | 7.56 | 0.81 | 647 | 1.1683 |
| | O | 9.37 | 8.44 | 0.90 | 703 | 1.2012 |



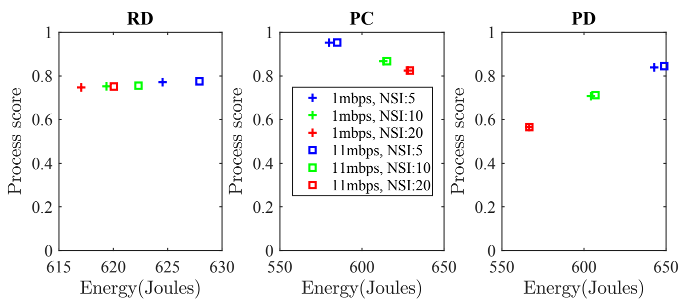
(a)



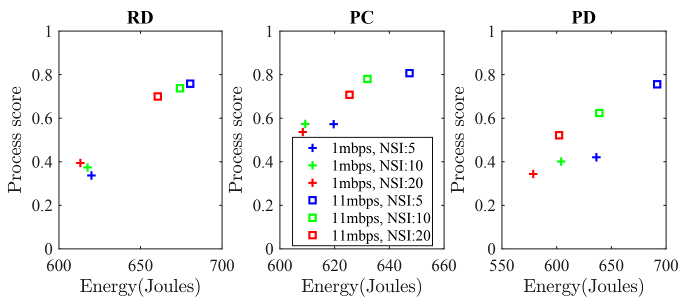
(b)

Fig. 12: Efficiency Scores (a) RWP (b) SAIVT.

The NSI frequency corresponds to how frequently nodes are updated with neighbour information and how often the proposed optimisation algorithms are executed. The cost of periodically executing optimisation algorithm was described in Section 3.6. The performance of the proposed algorithms increased when the NSI exchanges were frequent (from once every 20 seconds to once every 5 seconds). This signifies the importance of having recent NSI about neighbouring nodes. Particularly, PD was highly dependant on the frequency of NSI exchange. When the frequency was low (once every 20 seconds), it performed worse than the NO case, but when it was higher, the performance was better. The trend was consistent in both target datasets. For RD the NSI frequency rate should have no effect because it is asynchronous and nodes communicates with its neighbours when they seek help only. However, as seen in Fig. 13, there is some variation in performance, this is due to different sampling duration of NSI. For NSI $5, 10, 20$ second frequency, the moving average was calculated from the last $4, 9$ and $19$ seconds respectively. The opposite energy trends for the RWP dataset between PC and PD for various NSI frequencies also draw attention (Fig. 13a). However, upon further study, the energy usage was based more on CPU usage than on NSI exchanges.

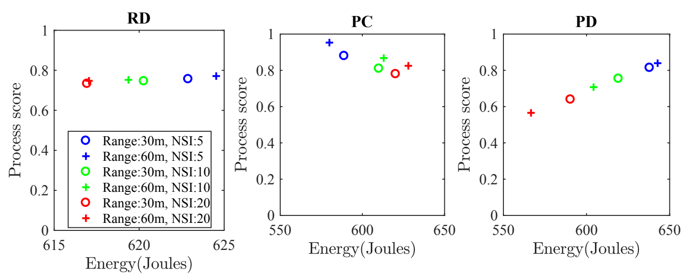


(a)

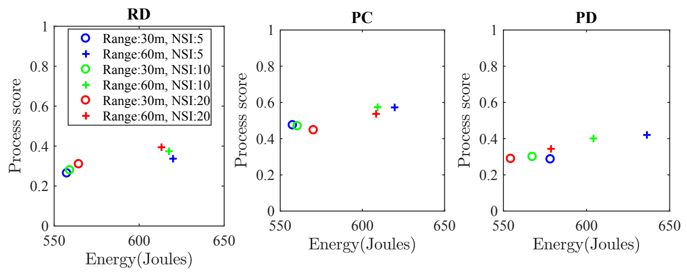


(b)

Fig. 13: Effect of communication bandwidth $(1, 11)$, and NSI frequency $(5, 10,$ and 30 seconds) (a) **RWP**: Performance increased as NSI update frequency increased, however, no significant difference as bandwidth increased. (b) **SAIVT**: Performance increased as the result of increased bandwidth and NSI update frequency.



(a)



(b)

Fig. 14: Effect of communication range $(30, 60$ metres) and NSI frequency $(5, 10,$ and 30 seconds). Slight improvement in performance as the range was extended except for PD in RWP case. (a) **RWP.** (b) **SAIVT**.

## 6.5 Effect of Communication Range

As the communication range of a node is increased, the number of neighbours the node can talk to increases (and vice-versa) – see Alg.2. The range was changed to see how the algorithms behave in varying conditions. Heuristically, more neighbours mean more options so the proposed algorithms should perform better when the communication range increases and vice-versa. The experiments generally follow this belief and the results are shown in Fig. 14. However, some interesting results were noted in the case of PD for the RWP case. The performance slightly reduced in this case when the communication range was extended for the lower frequency of NSI exchange (10 and 20). This is because as the NSI frequency was low and there were many neighbours, the uncertainty of their state was higher and led to decisions that were not optimal. However, the trend was not evident in the SAIVT case. In future works, more simulations will be carried out with different degrees of communication links to further investigate this behaviour.

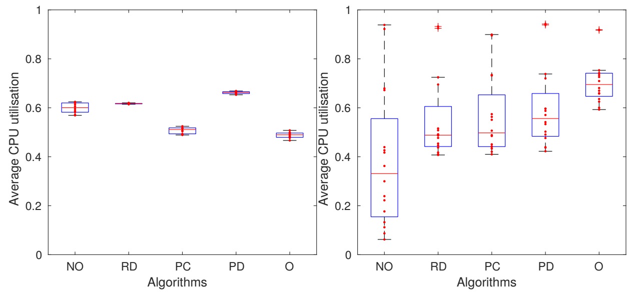
## 6.6 Average CPU Utilisation

The main idea behind the proposed algorithms is the distribution of the computational load among the nodes so as to minimise overloading as much as possible. Fig. 15 shows the average spread of CPU utilisation among the nodes. For RWP, the median CPU utilisation for PC and O across the nodes reduced by approximately 12 and 15% compared to the NO case, leading to reduced energy usage. In case of PD the median usage increased slightly be appoximately 6% while the RD the change was negligible. Due to the fact that the targets distribution were uniformly random and the resources usage is evenly distributed already, the performance gains were not large.

However, in the real dataset case, the overall CPU usage was higher and spread more evenly for the proposed algorithms than the NO case, which is signified by shorter boxes (see Fig. 15b). This led to significant performance gains meaning less targets were dropped. This may also lead to longer network lifetimes. The CPU usage in the NO case shows some sensor using three time more than the median and about nine time more than the sensor using lowest CPU. This would mean very short network lifetime, as the one using the most CPU would run out of battery sooner than the rest. In all the proposed algorithms, the median of average CPU usage is raised (signifying more performance) but bar some of the outliers, some of the sensors have reduced CPU usage which suggests network lifetimes may be extended.

## 6.7 Mean Execution Time

The simulation considered in this work is a soft real-time system. So a threshold was set for each every algorithm to be completed. The threshold was set to 30 and 20 seconds for RWP and SAIVT respectively. The Algorithm drop statistics corresponds to the algorithms that were not completed within the threshold period. Among those processed successfully, the mean execution times are compared. The results are shown in Table 9. The results show that even though offloading requires data to be offloaded, processed remotely and the results sent back to the offloader, the average execution time is comparable to the baseline NO



(a) (b)

Fig. 15: Average CPU utilisation across the nodes and NSI frequency (5, 10, and 30 seconds.) (a) RWP (b) SAIVT.

TABLE 9: Mean Execution times (Seconds)

| Dataset | NO | RD | PD | PC | O |
|---|---|---|---|---|---|
| RWP | 12.70 | 11.37 | 9.96 | 11.92 | 8.84 |
| SAIVT | 7.72 | 7.46 | 8.14 | 7.19 | 8.39 |

case and often better. The Oracle had the shortest execution time of all the algorithms tested including the baseline for the RWP dataset, whereas PC had the shortest time for the SAIVT case. The centralised algorithms performed better in this metric which could be because it considers all the neighbouring states and less likely to make wrong assumptions about neighbours.

## 7 Conclusion

In this paper, a sensor network was modelled as a network of queues using an Open Jackson network model, in the interest of computational load balancing in absence of cloud and fog. The network conditions were verified to be adequate using NS-3 simulations which showed latency in milli-seconds for worst case. Various novel *reactive* and *proactive* algorithms were proposed, which significantly enhanced the performance of the system compared to the Non-Offloading scenario. The algorithms were tested on Random Waypoint Model and a real SAIVT person re-identification dataset for different scenarios such as higher and lower bandwidth, higher and lower update rates etc. The results reinforce the assertion that most of the jobs can be processed if (a) the total job rate is less than total computing capability, and (b) if another node NSI is available. Especially in the real dataset, the performance improvements were significant. The performance boost also comes at similar energy cost and may well increase the network lifetime. This area of work has not been studied and explored before.

## 8 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. The authors would like to thank Jinho Kim for his help in NS-3 simulation. Simulation data will be made available.

## References

[1] X. Wang and R. Zhao, *Person Re-identification: System Design and Evaluation Overview*, pp. 351–370. London: Springer London, 2014.
[2] X. Wang, "Face Identification," *Computer Vision: A Reference Guide*, pp. 279–285, 2014.
[3] S. Sthapit, J. Thompson, J. R. Hopgood, and N. M. Robertson, "Distributed Implementation for Person Re-Identification," in *2015 Sensor Signal Processing for Defence (SSPD)*, pp. 1–5, sep 2015.
[4] S. Sthapit, J. R. Hopgood, N. M. Robertson, and J. Thompson, "Offloading to neighbouring nodes in smart camera network," in *2016 24th European Signal Processing Conference (EUSIPCO)*, pp. 1823–1827, aug 2016.
[5] K. Kumar and Y.-H. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?," *Computer (Long. Beach. Calif).*, vol. 43, no. 4, pp. 51–56, 2010.
[6] S. Yi, C. Li, and Q. Li, "A Survey of Fog Computing: Concepts, Applications and Issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, Mobidata '15, (New York, NY, USA), pp. 37–42, ACM, 2015.
[7] S. Sthapit, J. R. Hopgood, and J. Thompson, "Distributed computational load balancing for real-time applications," in *2017 25th European Signal Processing Conference (EUSIPCO)*, pp. 1189–1385, aug 2017.
[8] 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 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, (New York, NY, USA), pp. 49–62, ACM, 2010.
[9] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic Execution Between Mobile Device and Cloud," in *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, (New York, NY, USA), pp. 301–314, ACM, 2011.
[10] A. u. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A Survey of Mobile Cloud Computing Application Models," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 393–413, 2014.
[11] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A Survey of Computation Offloading for Mobile Systems," *Mobile Networks and Applications*, vol. 18, pp. 129–140, feb 2013.
[12] H. Wu, "Multi-objective decision-making for mobile cloud offloading: A survey," *IEEE Access*, vol. PP, no. 99, pp. 1–1, 2018.
[13] J.-y. Liu, E. Ahmed, M. Shiraz, A. Gani, R. Buyya, and A. Qureshi, "Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions," *J. Network and Computer Applications*, vol. 48, pp. 99–117, 2015.
[14] H. Wu, Y. Sun, and K. Wolter, "Analysis of the Energy-Response Time Tradeoff for Delayed Mobile Cloud Offloading," *SIGMETRICS Perform. Eval. Rev.*, vol. 43, pp. 33–35, sep 2015.
[15] Y. Zhang, D. Niyato, and P. Wang, "Offloading in Mobile Cloudlet Systems with Intermittent Connectivity," vol. 14, pp. 2516–2529, dec 2015.
[16] X. Chen, "Decentralized Computation Offloading Game for Mobile Cloud Computing," vol. 26, pp. 974–983, apr 2015.
[17] V. Cardellini, V. De Nitto Personé, V. Di Valerio, F. Facchinei, V. Grassi, F. Lo Presti, and V. Piccialli, "A Game-theoretic Approach to Computation Offloading in Mobile Cloud Computing," *Math. Program.*, vol. 157, pp. 421–449, jun 2016.
[18] Y. D. Lin, E. T. H. Chu, Y. C. Lai, and T. J. Huang, "Time-and-Energy-Aware Computation Offloading in Handheld Devices to Coprocessors and Clouds," vol. 9, pp. 393–405, jun 2015.
[19] C. M. S. Magurawalage, K. Yang, L. Hu, and J. Zhang, "Energy-efficient and network-aware offloading algorithm for mobile cloud computing," *Computer Networks*, vol. 74, no. Part B, pp. 22–33, 2014.
[20] D. T. Hoang, D. Niyato, and P. Wang, "Optimal admission control policy for mobile cloud computing hotspot with cloudlet," in *2012 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 3145–3149, April 2012.
[21] Z. Guan and T. Melodia, "The value of cooperation: Minimizing user costs in multi-broker mobile cloud computing networks," *IEEE Transactions on Cloud Computing*, vol. 5, pp. 780–791, Oct 2017.
[22] T. Truong-Huu, C. K. Tham, and D. Niyato, "A Stochastic Workload Distribution Approach for an Ad Hoc Mobile Cloud," in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pp. 174–181, dec 2014.
[23] S. Di and C. L. Wang, "Dynamic optimization of multiattribute resource allocation in self-organizing clouds," vol. 24, pp. 464–478, March 2013.

[24] W. J. Stewart, *Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling*. Princeton, NJ, USA: Princeton University Press, 2009.
[25] J. Vilaplana, F. Solsona, I. Teixidó, J. Mateo, F. Abella, and J. Rius, "A queuing theory model for cloud computing," *The Journal of Supercomputing*, vol. 69, pp. 492–507, jul 2014.
[26] C. Osorio and M. Bierlaire, "An analytic finite capacity queueing network model capturing the propagation of congestion and blocking," *European Journal of Operational Research*, vol. 196, no. 3, pp. 996 – 1007, 2009.
[27] S. Balsamo, V. D. N. Personè, and P. Inverardi, "A review on queueing network models with finite capacity queues for software architectures performance prediction," *Perform. Eval.*, vol. 51, pp. 269–288, Feb. 2003.
[28] Y. Takahashi, H. Miyahara, and T. Hasegawa, "An approximation method for open restricted queueing networks," *Oper. Res.*, vol. 28, May 1980.
[29] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.
[30] E. Meskar, T. D. Todd, D. Zhao, and G. Karakostas, "Energy Aware Offloading for Competing Users on a Shared Communication Channel," vol. 16, pp. 87–96, jan 2017.
[31] S. Biaz and S. Wu, "Rate adaptation algorithms for IEEE 802.11 networks: A survey and comparison," in *2008 IEEE Symposium on Computers and Communications*, pp. 130–136, jul 2008.
[32] J. Zhang, K. Tan, J. Zhao, H. Wu, and Y. Zhang, "A Practical SNR-Guided Rate Adaptation," in *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, apr 2008.
[33] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha, "DevScope: A Nonintrusive and Online Power Analysis Tool for Smartphone Hardware Components," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '12, (New York, NY, USA), pp. 353–362, ACM, 2012.
[34] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," Tech. Rep. January, 2008.
[35] W. Dargie, "A Stochastic Model for Estimating the Power Consumption of a Processor," vol. 64, pp. 1311–1322, may 2015.
[36] R. LiKamWa, B. Priyantha, M. Philipose, L. Zhong, and P. Bahl, "Energy Characterization and Optimization of Image Sensing Toward Continuous Mobile Vision," in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, (New York, NY, USA), pp. 69–82, ACM, 2013.
[37] F. Bai and A. Helmy, "A Survey of Mobility Models in Wireless Adhoc Networks," *Wireless Ad Hoc and Sensor Networks*, pp. 1–30, 2004.
[38] A. Bialkowski, S. Denman, S. Sridharan, C. Fookes, and P. Lucey, "A Database for Person Re-Identification in Multi-Camera Surveillance Networks," in *2012 International Conference on Digital Image Computing Techniques and Applications (DICTA)*, pp. 1–8, dec 2012.
[39] G. Carneiro, "Ns-3: Network simulator 3," in *UTM Lab Meeting April*, vol. 20, 2010.
[40] M. Lacage and T. R. Henderson, "Yet another network simulator," in *Proceeding from the 2006 Workshop on Ns-2: The IP Network Simulator*, WNS2 '06, (New York, NY, USA), ACM, 2006.
[41] H. Mao, S. Yao, T. Tang, B. Li, J. Yao, and Y. Wang, "Towards Real-Time Object Detection on Embedded Systems," *IEEE Transactions on Emerging Topics in Computing*, vol. PP, no. 99, p. 1, 2017.

**Saurav Sthapit** is currently a Ph.D. student in the Institute for Digital Communications, within the School of Engineering, at the University of Edinburgh, Scotland. He received his B.E. in Electronics and Communication Engineering from Tribhuvan University, Nepal and M.Sc. degree in Embedded Systems from the University of Kent, England. His research interests include computer vision, mobile computing and reinforcement learning etc.

**John Thompson** is currently a Professor at the School of Engineering in the University of Edinburgh. He specializes in antenna array processing, cooperative communications systems and energy efficient wireless communications. He has published in excess of three hundred papers on these topics. He is a work package leader for the EPSRC/DSTL Signal Processing for the Network Battlespace project. He is an editor for the Green Communications and Computing Series that appears regularly in IEEE Communications Magazine. In January 2016, he was elevated to Fellow of the IEEE for contributions to antenna arrays and multi-hop communications.

**Neil M. Robertson** (SM'10) received the M.Sci. degree from Glasgow University, Glasgow, U.K., in 2000, and the D.Phil. degree from Oxford University, Oxford, U.K., in 2006. He is Professor of Research for Image and Vision Systems at Queen's University of Belfast and coleads the EPSRC Edinburgh Centre for Robotics and the EPSRC/DSTL University Defence Research Centre. His work centres on human behavior recognition, computer vision and multi-modal sensor fusion with his research team. From 2000 to 2007, he worked in the U.K. Scientific Civil Service with DERA and held the 1851 Royal Commission Fellowship at Oxford University.

**James R. Hopgood** (M'02) is a Senior Lecturer in the Institute for Digital Communications, within the School of Engineering, at the University of Edinburgh, Scotland. He received the M.A., M.Eng. degree from the University of Cambridge in Electrical and Information Sciences in 1997, and a Ph.D. in July 2001 in Statistical Signal Processing, part of Information Engineering, also from the University of Cambridge.

He was then a Post-Doctoral Research Associate for one year in the Signal Processing Laboratory in the Cambridge University Engineering Department, and then a Research Fellow at Queens' College, Cambridge until March 2004. James joined the University of Edinburgh as a Lecturer in April 2004, and then Senior Lecturer in 2012. Since September 2011, he is Editor-in-Chief for the IET Journal of Signal Processing.

His research expertise include model-based Bayesian signal processing, speech and audio signal processing in adverse acoustic environments, including blind dereverberation and multi-target acoustic source localisation and tracking, single channel signal separation, distant speech recognition, audio-visual fusion, medical imaging, blind image deconvolution, and general statistical signal and image processing.