THE UNIVERSITY
of EDINBURGH

Computation Offloading for Algorithms in Absence of the Cloud

Saurav Sthapit

THE UNIVERSITY
of EDINBURGH

Thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy
to the
University of Edinburgh — 2017

Declaration

I declare that this thesis has been composed solely by myself and that it has not been submitted, either in whole or in part, in any previous application for a degree. Except where otherwise acknowledged, the work presented is entirely my own.

Saurav Sthapit

December 2017

Abstract

Mobile cloud computing is a way of delegating complex algorithms from a mobile device to the cloud to complete the tasks quickly and save energy on the mobile device. However, the cloud may not be available or suitable for helping all the time. For example, in a battlefield scenario, the cloud may not be reachable. This work considers neighbouring devices as alternatives to the cloud for offloading computation and presents three key contributions, namely a comprehensive investigation of the trade-off between computation and communication, Multi-Objective Optimisation based approach to offloading, and Queuing Theory based algorithms that present the benefits of offloading to neighbours.

Initially, the states of neighbouring devices are considered to be known and the decision of computation offloading is proposed as a multi-objective optimisation problem. Novel Pareto optimal solutions are proposed. The results on a simulated dataset show up to 30% increment in performance even when cloud computing is not available. However, information about the environment is seldom known completely. In Chapter 5, a realistic environment is considered such as delayed node state information and partially connected sensors. The network of sensors is modelled as a network of queues (Open Jackson network). The offloading problem is posed as minimum cost problem and solved using Linear solvers. In addition to the simulated dataset, the proposed solution is tested on a real computer vision dataset. The experiments on the random waypoint dataset showed up to 33% boost on performance whereas in the real dataset, exploiting the temporal

and spatial distribution of the targets, a significantly higher increment in performance is achieved.

Acknowledgements

First and foremost, I would like to thank my supervisors Dr James Hopgood and Prof. John Thompson for their excellent supervision throughout this project. I would like to say a big thank you for all the supervision and guidance. The meetings I had with you always had a calming effect during the years, when I am lost and panicking. To say I couldn't have completed this work without your exemplary supervision would be an understatement. Equally, special thanks go to Prof. Neil Robertson for lending his expertise in computer vision which is a substantial part of this thesis.

Next, I would like to thank my friends and colleagues in both Digital Communication lab at the University of Edinburgh and the Vision lab in the Heriot-Watt University. Even though most of us work in different domains, I always enjoyed the company and all the fun we had during these years.

I gratefully acknowledge the funding received towards my PhD studies from the School of Engineering and University Defence Research Consortium (UDRC).

I would like to thank my family: my parents, and my brother and sister in law for supporting me spiritually throughout this project and my life in general. I feel fortunate to have such wonderful parents. Last but not least, I am very grateful to my wife Anita, for encouraging me to do a PhD in the first place and then continuously supporting me for the entire length of the PhD. I am genuinely thankful for having you by my side every day.

Contents

List of Tables

List of Figures

Acronyms

1GSS first generation surveillance system. 34

2GSS second generation surveillance system. 34, 35

3GSS third generation surveillance system. xviii, 34, 35, 41

ADK Android Development Kit. 61

AODV Ad-hoc On-demand Distance Vector. 15, 29, 109, 124

AP Application Processor. 90

APM Active Power Management. 94, 98, 99, 152

BER Bit Error Rate. xxiii, 66, 110, 111

BiCov Bio-inspired Covariance based features. 44, 45

BW Network Bandwidth. 13, 74, 78–81, 97, 110, 113, 137, 140

CCTV Closed Circuit Television. xvii, 5, 34

CMC Cumulative Matching Characteristics. 55, 58

CNN Convolutional Neural Network. 52

COTS commercial off-the-shelf. 42, 89, 93, 105

CPU Central Processing Unit. xiii, xix–xxi, xxiv, 36, 60, 72, 76–80, 84, 89–91, 94, 97, 98, 106, 109, 110, 116, 118, 119, 121, 124, 133–136, 145, 149, 153

CSMA Carrier Sense Multiple Access. 13

CV Computer Vision. 34, 36, 50

DCH Dedicated Channel. xix, 91–93

DNN Deep Neural Network. xii, 19, 50–53, 56, 59, 60, 68

DSDV Destination-Sequenced Distance-Vector. 14, 15, 29, 109

DSR Dynamic Source Routing. 15, 124

DSTL Defence Science and Technology Laboratory. 3

DUT Device Under Test. 89, 93, 94

dv *decision vector*. 125

DVFS Dynamic Voltage and Frequency Scaling. 94

ES Efficiency Score. xii, 96, 99, 103, 131, 152

FACH Forward Access Channel. 91, 92

FCFS First Come First Service. 23, 26

FOB forward operating base. 39

FOV Field Of View. xviii, xix, 36, 39, 42, 71, 72, 82, 83, 96, 98, 126

FPGA Field Programmable Gate Array. 79, 107

FPS Frames Per Second. 60, 73, 89–91, 93, 94, 126

FSM Finite State Machine. 66

GAN Generative Adversarial Networks. 50

GPU Graphical Processing Unit. 36, 73, 79, 107, 145

HOG Histogram of Gradients. xi, xvii, 19–22, 37, 38

HSUPA High-Speed Uplink Packet Access. 63

HSV Hue Saturation Value. 45, 57

IC Integrated Circuit. 1

IOT Internet Of Thing. 106, 139, 141

ITML Information Theoretic Metric Learning. 44–46

JVM Java Virtual Machine. 56

KISSME Keep It Simple and Straightforward MEtric. 44–46, 55, 57, 59, 139

k-NN k-Nearest Neighbour. 21

LAFT Locally Aligned Feature Transform. 44–46

LBP Local Binary Pattern. 20, 38, 45, 57

LDML Logistic Discriminant Metric Learning. 46

MANET Mobile Ad-hoc Network. xi, xvii, xviii, 6, 9–16, 27, 28, 109

MARS Motion Analysis and Re-identification Set. 54, 55

MBI Minimal Battery Impact. xii, 86, 87, 99, 101, 103, 105, 137, 152

MC Markov Chain. 23

MCC Mobile Cloud Computing. xvii, 9, 10, 73, 75, 79, 84, 107, 117

MDP Markov Decision Process. 108

MEC Minimal Energy Cost. xii, 85–87, 99, 101, 103, 105, 137, 140, 152

MIPSA Modular Integrated Passenger Surveillance Architecture. xviii, 35, 41

MOG Mixture of Gaussians. 77

MSCR Maximally Stable Color Region. 21, 45, 47, 57, 58

NN Neural Network. 21, 49

NO Non Offloading. xx, xxi, 96, 99, 101–103, 105, 120, 129–131, 133–137, 152

NSI Node State Information. xv, xx, xxi, 107, 110, 112, 116, 117, 120–125, 128–130, 132–138, 140, 141

O Oracle. xiii, 120, 121, 129, 131, 133, 136

OLSR Optimised Link State Routing. 14

OOB Offload Only if Busy. xii, 88, 99, 101, 103, 105, 140, 152

OS Operating System. xv, 64, 89, 91, 95

PC Proactive Centralised. xiii, 120, 121, 123, 124, 129–131, 133, 136, 137, 140, 141

PCA Principal Component Analysis. 45, 57, 59

PD Proactive Distributed. xiii, xxi, 120, 123, 129–133, 135, 140, 141

pdf probability distribution function. 46

PDR Packet Delivery Rate. xx, 66, 110–112, 122, 124, 128

PPP Poisson Point Process. 17, 96

PRID Person Re-identification. xii, xv, xviii, 6, 30–32, 36–45, 50–55, 60, 67, 69, 71, 72, 74, 80, 82–84, 96, 105, 139

RAM Random Access Memory. 56, 61

RD Reactive Distributed. xiii, 120, 124, 129–131, 133, 137, 138, 140

ReLU Rectified Linear Unit. 52

RFH Request For Help. 124, 125, 138

RGB Red Green Blue. 34

RHSP Recurrent High-Structured Patches. xviii, 45, 47, 57, 58

RNC Radio Network Controller. 92

RRC Radio Resource Control. 92

RSSI Received Signal Strength Indication. 63, 66

RTT Round Trip Time. 74, 75, 107

RWP Random Waypoint Model. xiii, xvii, xx, xxi, 15–18, 124, 126, 128, 129, 131–137

SDALF Symmetry-Driven Accumulation of Local Features. xii, xv, xviii, 44, 47, 57–59

SDC Saliency guided Dense Correspondence. 49

SIFT Scale Invariant Feature Transform. xxiv, 20, 38, 45, 48

SNR Signal to Noise Ratio. 110, 111

SVM Support Vector Machine. xvii, 20–22

TORA Temporarily Ordered Routing Algorithm. 15

UE User Equipment. 66, 91, 92, 124

VANET Vehicle Ad-hoc Network. 140

VIPeR Viewpoint Invariant Pedestrian Recognition. xv, 51, 53, 55–58

W-CDMA Wideband-Code Division Multiple Access. 63

WHSV Weighted Color Histograms. 47

XML Extensible Markup Language. 126

Definitions

app is an application program, generally developed for deployment in a Smartphone. 56, 59, 61, 145

Computation Offloading is a process of offloading computationally intensive algorithm from one computing facility to another aiming to save time, energy or both. xix, 6, 30–32, 43, 69, 71, 72, 74–76, 78, 79, 82, 83, 99, 102, 103, 105, 139, 141, 142

$M/M/1$ A simple queue with one server, Poisson arrival, exponential service times, and infinite queue length. xi, xviii, 23–25, 108–110

offloader The device that sends their job to cloud or neighbour for it to be executed and result sent back. xix, 75, 77–83, 87

onloader The device that takes on other's job for execution and sends result back after execution. xii, xix, xxiii, 75, 77–83, 87, 101, 108

Chapter 1

Introduction

This project explores neighbouring devices as an alternative to the cloud for computation in *Computation Offloading* problems. This chapter briefly introduces *Computation Offloading*, why it could be useful and when the traditional offloading to cloud isn't feasible. Lastly, the chapter also presents the layout of the thesis.

1.1 A Short Overview

Until recently, a common characteristics of the computers was that all of them were fixed in one location. Lately however, these desktops have been partially substituted with mobile technologies such as laptops and smartphones with comparable computing capabilities. Crucially, they also have wireless communication capabilities added which allows them to connect to one another and the main *grid* such as the Internet, from anywhere within the wireless range. Likewise, in the software domain, simple computer programs have been replaced with intelligent and complex algorithms. For example, instead of the user having to enter the commands manually, artificially intelligent assistants such as *Ok Google* [1], *Apple's Siri* [2], and *Amazon's Alexa* [3] exist, that can decipher human commands and can carry out instructions.

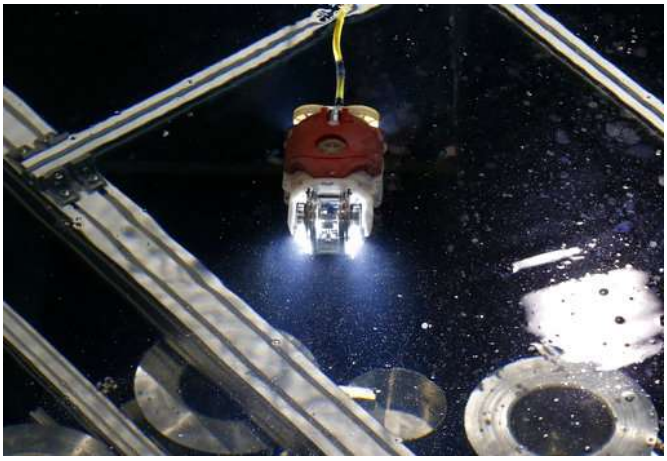


Figure 1.1: A newly developed robot to probe the hazardous underwater nuclear site. Figure from [5]

In lieu of the hardware and software development, computing platforms are more distributed, dynamic, and intelligent. They can be deployed in the field for specialised and complicated tasks. For example, robots can be deployed to collect sensor data and work in hazardous conditions where it is dangerous for humans to go. In 2011 Fukushima, Japan, after a tsunami, nuclear plants were too hostile an environment for humans to go and assess. Robots that can be controlled from outside the contaminated area were built to assess the situation and also clean up underwater [4, 5]. Figure 1.1 shows a robot used to probe the environment underwater.

Even though there has been rapid progress in both hardware and software, mobile or embedded devices are still resource limited. They may lack the computational power of a desktop computer, storage media, and most importantly energy resources. As they are often battery powered, the lifetime of these devices may be limited. Once the energy resource is emptied, they are switched off and cannot function until the batteries are replenished. Also, it is often required for more than one device to collaborate with each other to complete a task. Those devices may be specially designed and deployed to work together or the sensors may be in vicinity by chance. In theory, similar to how the animals achieve synergy by working together, these robots and drones should be able to communicate and help each other to achieve the main objective. However, there are many obstacles in practice. The deployment devices are resource limited so

have limited computational capability and limited energy. While many tasks can be delay tolerant in nature, others are not. For example, sending an email or backing up data can be considered delay tolerant and can be done in a time window of say an 10 minutes, whereas identifying a person in the image frame has to be completed in a certain time-frame, e.g. 30 seconds. One way of achieving this time target is by upgrading the hardware or sacrificing accuracy to make the algorithm run faster. Another approach would be to offload the algorithm for execution on another device. In this thesis, the offloading approach is explored.

1.2 Computation Offloading

Instead of executing a computationally intensive algorithm on its own, sometimes it may make sense concerning time or energy savings, to send the data to another computing facility for computation. This process is called *Computation Offloading*. Generally, the computationally intensive algorithms are offloaded to the cloud with substantial computing resources. However, there are cases when the cloud may not be available such as underwater or underground scenarios. One such case is the battlespace which is described next.

1.3 Motivating Scenario – Battlespace

According to Defence Science and Technology Laboratory (DSTL), the ground soldier's uniform in future would be covered with number of sensors that can help them in the battlefield [6, 7]. Figure 1.2 shows the design of DSTL and the number of sensors embedded to soldier's uniform to materialise by 2025. The design includes [6]:

- *"Smart glasses concept which include a heads-up display, integrated camera and bone conducting headphones to increase situational awareness without compromising hearing."*

- *"A robust personal role computer concept enabling better information sharing and communications between personnel."*

Let the mission of the be to detect all the people in the scene and try to identify them. For that purpose, a group of soldiers and drones are deployed. The visual footage can come from the camera on the soldiers head or the drones that are deployed. If a criminal or an enemy is identified in the scene then one of the soldier can go and interrogate them. Each camera only reports to the soldier if a known criminal or enemy is identified. In this example, say one of the drone's camera has a higher number of humans detected –see Figure 1.3. In this image frame there are six people detected. Identifying all six of them in real-time is challenging. However, because the drone's processing capability is limited, it struggles to process all the targets detected in real time. Traditionally, the drone could ask the cloud for help in processing the targets. However, the cloud may not be available or the latency may be too high to ask for help. However, the personal role computer belonging to each soldier has in comparison higher processing power and higher battery capacity (backpack). Would it be possible for the drone to transfer the targets to one or more soldiers in order to

1. complete identifying people on allocated time,
2. extend the lifetime of the drone.

This could be particularly useful, if some sensors are busier than others or have less resources than others and access to cloud computing is not available. The work presented in this thesis can be applied in non battlespace scenarios as well such as underwater or underground where the cloud is not reachable.

1.4 Research Objectives

This thesis explores the communication capability of embedded devices to complete the computationally intensive jobs on time. It attempts to share the jobs among the

Figure 1.2: Future uniform of soldiers containing various intelligent sensors including a head mounted camera to help them in the battle. Figure from [6]

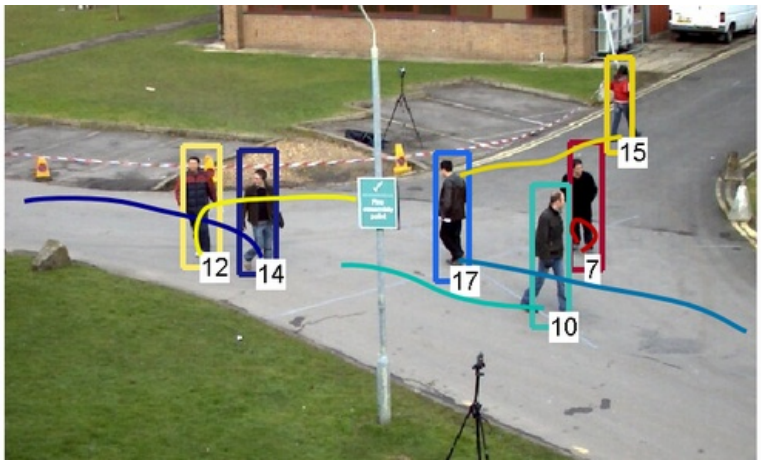


Figure 1.3: Image of Closed Circuit Television (CCTV) cameras used for detection and tracking of humans. The bounding boxes is the person's detection in the frame and path is the track of particular target along the frames. Image from [8]

deployed sensors in an energy efficient fashion to achieve better efficiency and better work completion rate. Based on the motivating example in the last section, the broad research objective of this thesis can be formulated as below. However, it is important to note that, it is not limited to computer vision algorithms nor specific hardware such as drones.

1. How can signal processing be leveraged in multiple devices with communication capabilities?
2. Would it be feasible for computationally intensive algorithms to be executed on embedded devices? If yes, what are the different costs of executing them on the resource limited devices?
3. Mobile devices utilise the cloud services for executing computationally intensive algorithms. Is this the best option even if the communication bandwidth is limited?
4. Could the network of embedded devices work in tandem in an energy efficient manner to improve the productivity of the overall group?

1.5 Thesis Structure and Research Contributions

The rest of thesis is structured in the following way.

Chapter 2 is the background chapter comprised of three different sections. The first section introduces the basic concepts on Mobile Ad-hoc Networks (MANETs), their communication and mobility models. Second section is on surveillance and computer vision algorithms. Finally, the last section is on queuing theory and network of queues. This chapter helps the reader to understand rest of the chapters.

Chapter 3 assesses factors affecting algorithm implementation on resource limited devices and various cost involved. Experiments are conducted out to cost of executing algorithms on the embedded devices such as an Android smartphone compared to communicating

data for offloading. The experiments carried out in this chapter were presented in the Sensor Signal Processing in Defence conference in 2015 [9].

In Chapter 4, presents a simple algorithm for *Computation Offloading*. It is a proof of concept that *Computation Offloading* can be useful even when the cloud is not available. In order to simulate the time and energy consumption on the mobile devices while performing computation and communication, a simulator was developed and described here. The simulator and the experiments carried out in this chapter were presented in European Signal Processing Conference in 2016 [10]

Chapter 5 presents a model of network of sensors as network of queues and a realistic environment is considered including the cost of optimisation, partially connected sensors etc. The results of this chapters were presented in European Signal Processing Conference in 2017 [11]. Also, a journal is under review in IEEE transactions in Mobile Computing.

Chapter 6 is the last chapter of the thesis. It discusses the main conclusions of the overall research carried out during the course of the PhD. It also proposes the future works that should be carried out in order to progress the work further is discussed.

Chapter 2

Background

The primary aim of this project is to offload computationally intensive algorithms to increase the performance of a network of embedded devices. This is referred as Mobile Cloud Computing (MCC) and contains three vital components; device platforms, algorithms and communications as shown in Figure 2.1. MCC is discussed in more detail in Chapter 4 however, this chapter covers the basics of these components. In the first part, computer vision theory and techniques will be explained. Computer vision algorithms in this context are considered to be computationally intensive algorithms that benefit from offloading or sharing. In the second part, a Mobile Ad-hoc Network (MANET) will be studied. Devices in MANETs by its nature are resource limited. So, they would benefit the most from MCC. In the last part, queueing theory and network flows will be discussed which is used in Chapter 5 to formulate the model of a sensor network and solve them.

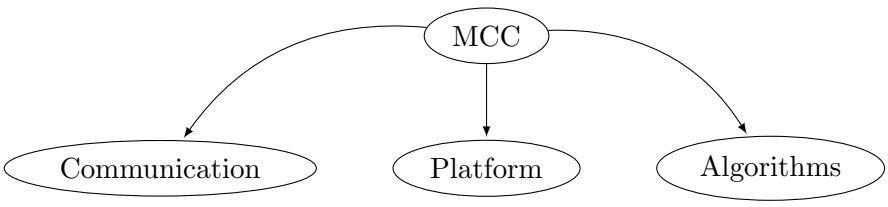


Figure 2.1: Three components of MCC

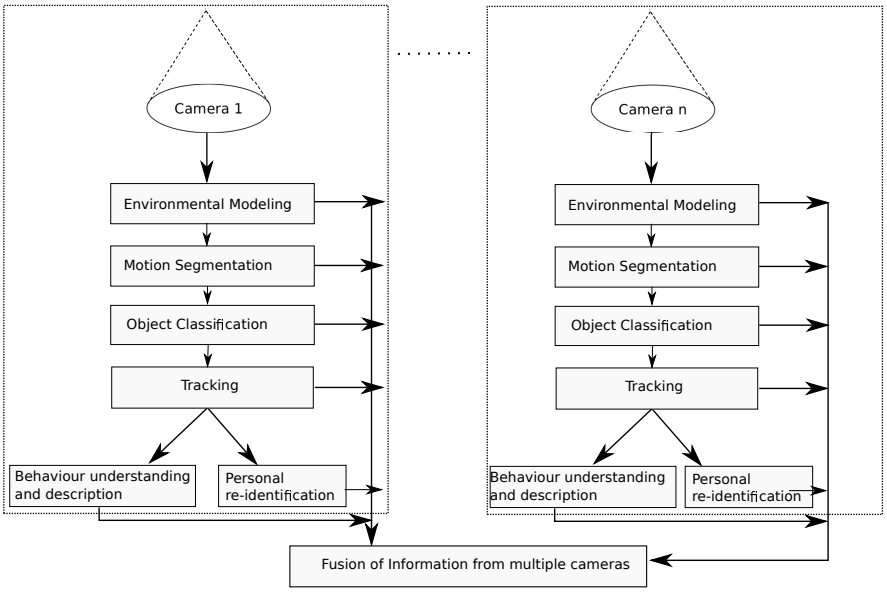


Figure 2.2: General visual surveillance framework as described by [12].

2.1 Surveillance Systems

Surveillance systems refer to monitoring target activities such as human or vehicle activities at certain times and locations, to improve safety and security in the area. Visual camera systems are one of the most commonly used modalities in surveillance systems owing to its rich information content. Generally speaking, the visual modality is omnipresent and sometimes complemented with other modalities such as audio, infra-red or olfactory signals. So in this section, visual surveillance systems will be briefly discussed, focussing on their evolution over time. This work is presented to help in understanding the direction that the research in surveillance systems is heading towards, and explain the significance of Person Re-identification (PRID) and *Computation Offloading*, in the context of the security and surveillance systems.

A standard framework for a visual surveillance system starts with capturing images using cameras, which are then fed to various computer vision algorithms. These algorithms may include environment modelling, motion segmentation, object classification, tracking, identification and activity recognition [12–16]. For example, each block in the software

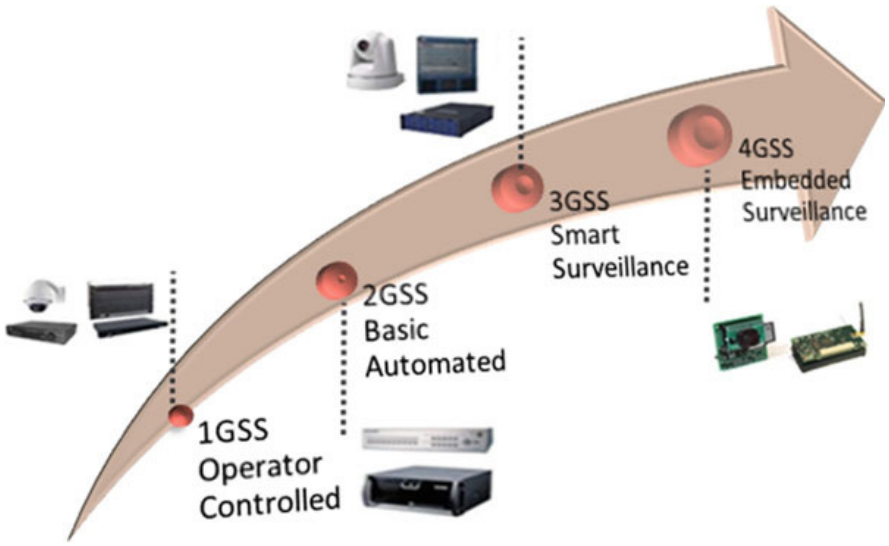


Figure 2.3: Surveillance systems evolution as described by [17]. Image from [17]

framework [12] shown in the Figure 2.2 represents one of these algorithms and directly or indirectly helps the system extract some vital information about the target activity.

2.1.1 Evolution of Surveillance Systems

Discussing the generations and history of surveillance systems helps one to understand how it has matured over the years and in which direction the research is heading. Marcenaro et al. [13] first started the notion of generations in the surveillance systems. Even though there are no clear boundaries, researchers such as [13, 15, 17] have a consensus on the salient features of each generation. Today, the technology has matured to the fourth generation – see Figure 2.3 [17]. All the generations will be briefly discussed here with more focus on the later generations.

The first generation was characterised by the use of analogue and wire-based technologies. Analog Closed Circuit Television (CCTV) cameras were used to capture images, which were then sent into a control room and recorded on analogue technologies such as videotapes. One or more operators in the control room need to monitor the video-feeds continuously, to find anomalies or a breach of security. As the system was based entirely

on analogue technologies, the signals were not encrypted at all, resulting in reduced security. Being analogue was one of the reasons why the first generation surveillance system (1GSS) could not be wireless as anyone could intercept and interfere with it. Computer Vision (CV) algorithms were not used at all.

In the second generation surveillance system (2GSS), the analogue feeds were digitised and stored on devices such as hard disk drives. More importantly, it allowed the application of basic CV algorithms for detection and tracking of targets. This increased the overall efficiency of the system. However, there were two noticeable problems with the 2GSS. First, the detection and tracking algorithms were not robust enough and, second, scaling the system to a large number of sensors was difficult due to centralised processing. With the advancement in electronics and computer vision, some of the issues were dealt with in the third generation surveillance system (3GSS), which allowed for smart and wide area surveillance systems. In addition to the Red Green Blue (RGB) sensor data, different modalities such as audio inputs were added which resulted in the more robust detection and tracking. Similarly, some of the work was also delegated to the camera or individual dedicated devices which allowed more freedom to the central processor. In turn, it allowed for broader area surveillance systems.

PRISMATICA is an excellent example of 3GSS that was implemented for smart surveillance in public transport [15]. The system was deployed in various stations in Paris, Milan and London. It comprised of visual camera inputs, audio inputs and inputs from contactless smart card devices as shown in Figure 2.4. In this system, the CCTV inputs were fed into a video matrix whereby operators could view it, in addition to it being digitised. Each video input was processed by individual video devices to detect targets and track them. The processed data was then stored in the database via the device network. Figure 2.4 also shows other technologies such as contactless smart cards and audio surveillance devices that were connected to the system via the device network. The overall system was controlled by the central entity called Modular Integrated Passenger Surveillance Architecture (MIPSA). In addition to gathering and processing the data from the sensors, the operators could control the cameras based on the data. For example, the system could detect any suspicious noise was picked up by a

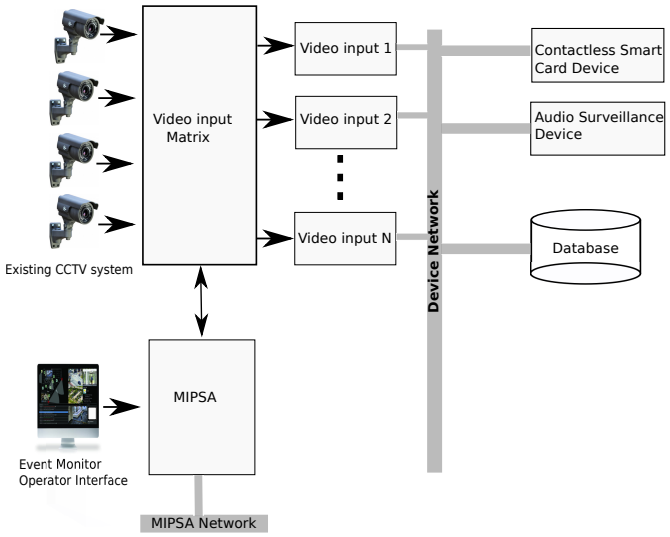


Figure 2.4: PRISMATICA: an example of 3GSS showing various components such as Modular Integrated Passenger Surveillance Architecture (MIPSA), network, database, and camera and other sensors. [15].

microphone using the audio activity detector, such as people shouting. Then the system could display the camera feed from the relevant localised area in the station.

The PRISMATICA system shows that the 3GSS was relatively smarter than 2GSS and could handle a large number of cameras. However, there are many limitations of this system. For example, all the cameras are connected using wires between the device network and the MIPSA which is the central entity. Failure of the MIPSA would be catastrophic and the whole system would fail as a result. Also, while there is a provision to extend the system to other stations by connecting via the MIPSA network, more extensive deployment would be expensive.

In the fourth generation, the focus started shifting towards distributed embedded devices with the wireless communication capabilities, which can be easily deployed and scalable yet smart enough to execute robust CV algorithms [17]. Such a system should be robust and highly fault tolerant. Apparently, the growth in hardware technologies has in some ways helped in achieving this. For example, over the last two decades, the interest in smart-phones has fuelled tremendous development in mobile processors that

are powerful, efficient and inexpensive. Similarly, pocket sized embedded devices such as the Raspberry Pi [18] and the NVidia Jetson Tk1 [19] have come to market that boasts multi-core Central Processing Unit (CPU) and multi-core Graphical Processing Unit (GPU), while maintaining their small form-factor and low cost. However, there are still many open problems associated with distributed sensor networks for surveillance systems. Obviously, there are implementation challenges, that will be discussed in Section 2.3.2. Apart from that, there are issues related to synchronisation in an ad-hoc network, having consensus between the sensors and network coverage etc.

In the next section, an essential CV algorithm is described in detail with examples to highlight the complexity and challenges concerning implementation for real-time surveillance systems.

2.2 Computer Vision Algorithms

Computer vision is an interdisciplinary field that seeks to automate tasks that a human visual system can perform [20,21]. The tasks may include acquiring, processing, analysing, and understanding visual images. They have wide range of applications. For example, they can be used in biomedical imaging to find tumour in the body; in security for detecting and monitoring anomalies; autonomous control etc. To understand the high dimensional data in the image data produced by the image sensor, features are used.

2.2.1 Feature Extraction

According to Gonzalez and Woods, there is no formal definition of what constitutes an *image feature* [22]. However, a feature is generally thought of as a distinctive attribute or description of "something" that is desired to be labelled or differentiated. Feature extraction constitutes of two principal aspects: *feature detection* and *feature description*. *Features detection* refers to finding the features in an image, region, or boundary. *Feature description* assigns quantitative attributes to the detected features. There are many

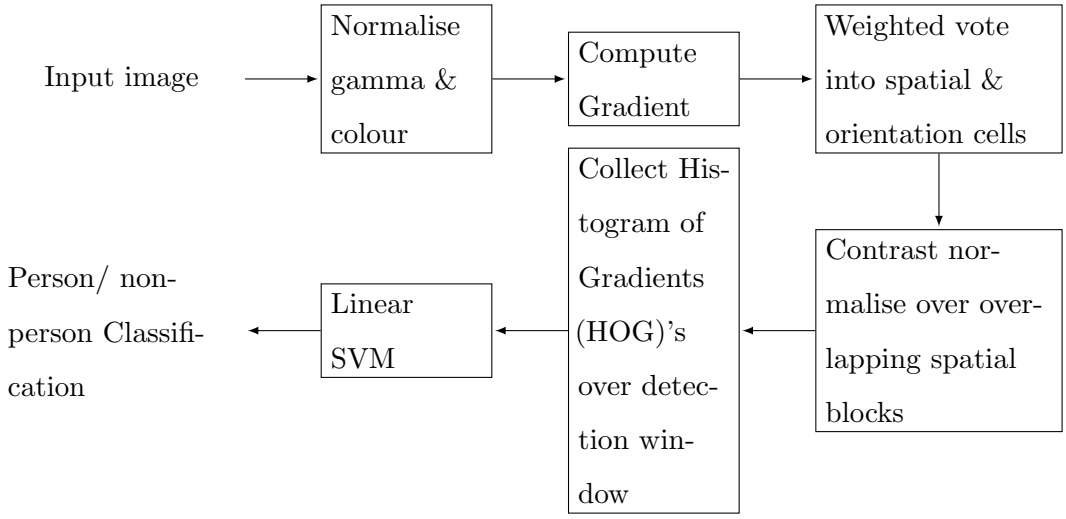


Figure 2.5: Flowchart for pedestrian detection based on HOG features [23]

types of features that are used in computer vision. For example, it could range from a histogram of the whole image to a computationally intensive feature learned via Deep Neural Network (DNN). The purpose of this section is not review all the features that are present in the literature but show how visual features are extracted and what kind of processing it entails. As an example, a person detection algorithm is described next.

2.2.2 Histogram of Gradients

The HOG features were used by Dalal and Triggs for person detection [23]. Person detection algorithms are used for exemplars in Chapters 3 to 5. The HOG based person detection has two distinct parts. The first part consists of extracting the features from an image that is fed to the second part that consists of Support Vector Machine (SVM) classifier. The process of extracting HOG features is depicted in Figure 2.5 and are described below

1. **Gradient Computation**

   The first major step is to calculate the gradients in the x and y directions. This is accomplished by convolving the kernels $\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$ and $\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}^T$ across the whole image. The gradients are used to find edges in the image.

2. **Orientation Binning**

   An image or a detection window is divided into small spatial regions called "cell" which is 8×8 pixels each. The pixel gradients calculated in step 1 is used for histogram creation for each of these cells. For detecting humans, Dalal used nine equally spaced bins in the range $0° - 180°$. So the histogram feature of each cell is nine dimensions.

3. **Contrast Normalisation**

   Four neighbouring "cells" (2×2) are called a "block". A block feature is the concatenation of features from four constituent "cells". See Figure 2.6 to compare "pixel", "cell" and "block". The block features are normalised over the block using L2Hys [23] which is known as contrast normalisation. For each image or detection window, the histogram features of overlapping blocks extracted. The overlapping is 50% (ie. 8 pixels). A test image and its HOG descriptor is shown in Figures 2.7d and 2.7e respectively.

To determine if there is a human in the image or not, a classifier such as SVM is used. The HOG features of human and non human images and their labels are used to train the SVM classifier. Figures 2.7b, 2.7c, 2.7f and 2.7g show various positive and negative weights for the training and the test images.

The HOG algorithm is described here to gently illustrate how local features of an image is extracted and used for something useful (in this case person detection). There are many other different types of features; Local Binary Pattern (LBP) [24], Scale Invariant Feature Transform (SIFT) [25], and Maximally Stable Color Region (MSCR) [26] to name a few. Similarly, there are numerous ways to compare or match them in a supervised and unsupervised manner. It could be as simple as finding Euclidean distance between them to more complex supervised methods like k-Nearest Neighbour (k-NN), SVM and Neural Network (NN) [27].

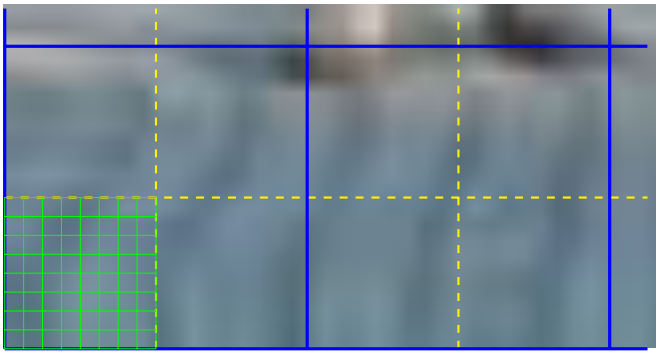


Figure 2.6: Bottom left of the sample image in Figure 2.7d showing size of pixel, "cell" and "block" for HOG feature extraction. Smallest square in green grid represents the pixels. A "Cell" is a patch of 8×8 pixels and are separated by the dashed yellow lines. A "Block" is 2×2 cells and separated by solid blue lines. For clarity, overlapping of "blocks" is not shown here.

2.3 Person Re-identification

Person Re-identification (PRID) refers to associating people across camera views at different locations and times [29]. It can have a considerable impact on surveillance and security applications like PRISMATICA because manual identification is not only tedious and costly, but the latency may be unacceptable. As the Field Of View (FOV) of the cameras can be non-overlapping, the background and pose can change, and the target can be occluded which makes PRID challenging. A particular individual can look dissimilar in different views, while different individuals can look similar from different angles. Figure 2.8 shows some sample pedestrian images from the VIPeR dataset [28] taken by two cameras. The posture of the pedestrians in the images of the two cameras are different even though they are walking in both cases. Additionally, in the first pair Figures 2.8a and 2.8h the trousers appear to be of different colour. This difference in colour of the same object in different images is called colour constancy problem [30]. In second pair Figures 2.8b and 2.8i, the camera 2 view shows an artefact (handbag) which is barely visible in the camera 1 image. Similarly, camera 1 images Figures 2.8d and 2.8g show front view of the pedestrians while the images of same individuals in

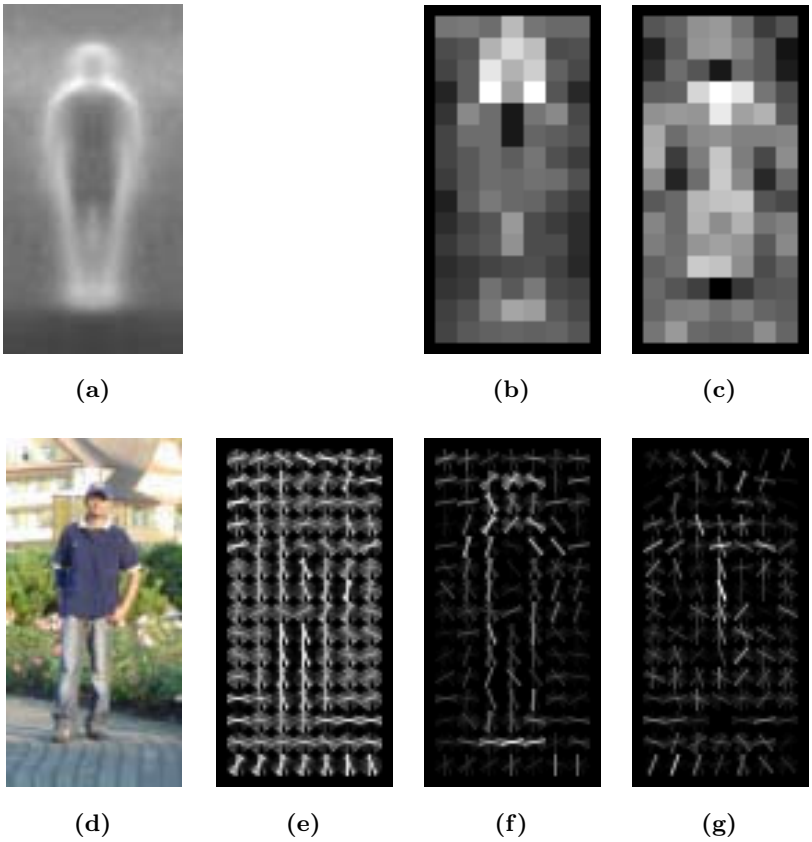


Figure 2.7: HOG processing pipeline. (a) Average gradient over the training samples (b) Maximum positive SVM weights in each block (c) Maximum negative SVM weights in each block (d) Test Image (e) HOG descriptor for the test image. (f) Positive weights for the test image. (g) Negative weights for the test image.

camera 2 show side views Figures 2.8k and 2.8n along with blurring further complicating the problem.

PRID algorithms follow the basic workflow depicted in Figure 2.9. First, images are taken from each camera and preprocessed. The pre-processing step may include background subtraction and a HOG based person detection algorithm described in Section 2.2. Features are then extracted from a person's image to form a unique signature. Popular features include a combination of low level features such as colour histograms, LBP [24], SIFT [25] and the HOG features [31]. For verification, a metric distance between

Figure 2.8: Seven pairs of pedestrian images from the VIPeR dataset [28]. Each column shows images of an individual. Images of same person from Camera 1 (first row) may appear significantly different in Camera 2 (second row)

signatures is calculated and compared to a threshold, to verify if the images belong to the same individual or not. Similarly, for identification, the probe signature is compared with the gallery set containing signatures of a seen individuals to find the closest match. Some researchers have also defined the person identification problem as a ranking problem such that the closes match appears at the top of the list and the least likely match appears at the bottom of the list [32].

2.3.1 Applications

The computational complexity of PRID algorithms will be analysed in detail later, but for now, let's look at two simple examples to understand the application and the implementation challenges in practical systems.

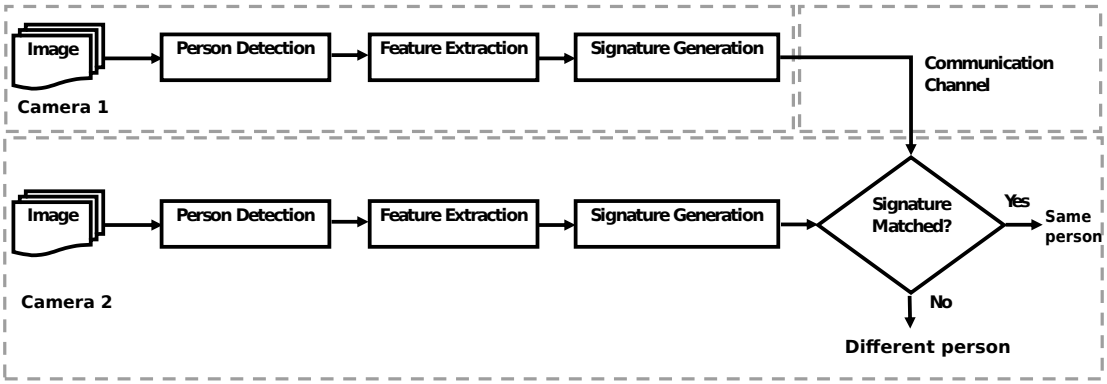


Figure 2.9: Decentralised Person Re-identification workflow. The centralised system would be exactly alike except for the absence of communication channel in the centralised system.

Case 1: Camera Handover

Consider a vision based pedestrian tracking system like the one described for PRISMATICA, which comprises of multiple smart cameras. The cameras may be fixed or moving, represented by dark and light camera icons respectively –Figure 2.10. The system could be implemented in a transportation area such as a train station, or more sensitive area such as a forward operating base (FOB) or an embassy of a country. The problem is that the FOV may be non-overlapping; however, the cameras are connected to each other via wireless links such as WiFi, Bluetooth or cellular. Targets 1 and 2 are moving along the path shown by the arrows. Inside a camera's FOV, targets can be tracked using various filters such as the Kalman filter [33] and Particle filter [34]. However, the targets often move from one camera's FOV to another. For the system to seamlessly track a person across cameras, the system must verify if the images from both cameras are of the same person or not. One way to assert that is using a PRID algorithm.

Case 2: Anomaly Detection and Active Surveillance

Let us consider the pedestrian tracking algorithms further using the PETS2014 dataset [35]. The dataset consists of four non-overlapping cameras mounted on each side of

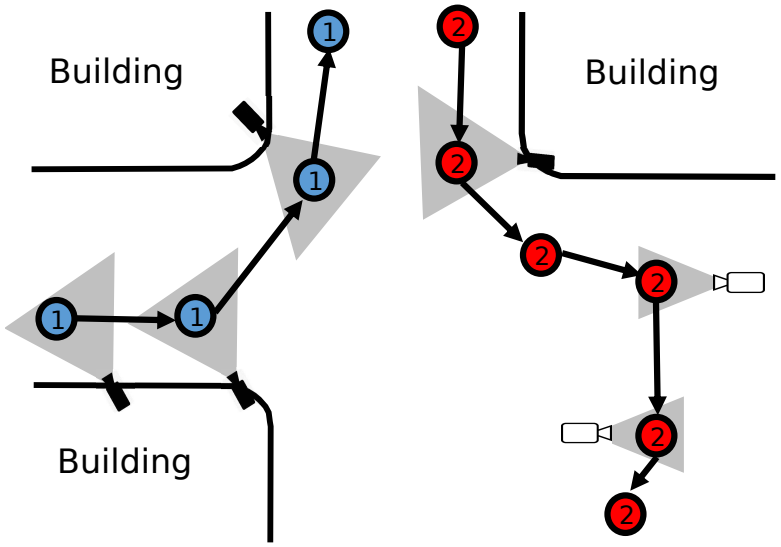


Figure 2.10: Application of multi-camera person re-identification in an outdoor scenario. Shaded cameras are fixed, white cameras are moving and grey area represent their FOV

a truck. In this dataset, some of the footage contains criminal activities such as a person stealing from the truck or people fighting with with each other. For example, Figure 2.11 shows a fighting incident with four people involved. First, the driver of the truck gets involved in a fight with one person; then two more people join the fight – Figure 2.11b. As the driver falls on the ground (Figure 2.11f), the three men run away – Figure 2.11e. The whole incident takes place in less than a minute. This incident could also be compared to someone planting a bomb in a busy area and walking away. It is highly desirable to detect such anomalous incidents, alert the security system and quickly identify the perpetrators to prevent further damage or injury. Robust PRID algorithms are required for these cases.

2.3.2 Implementation Challenges

Let us assume the PRID algorithm is robust and can perform with 100% reliability. There are three ways to implement them in real life – see Figure 2.12. In a centralised system like 3GSS, all the sensor nodes would be connected to a single server such as

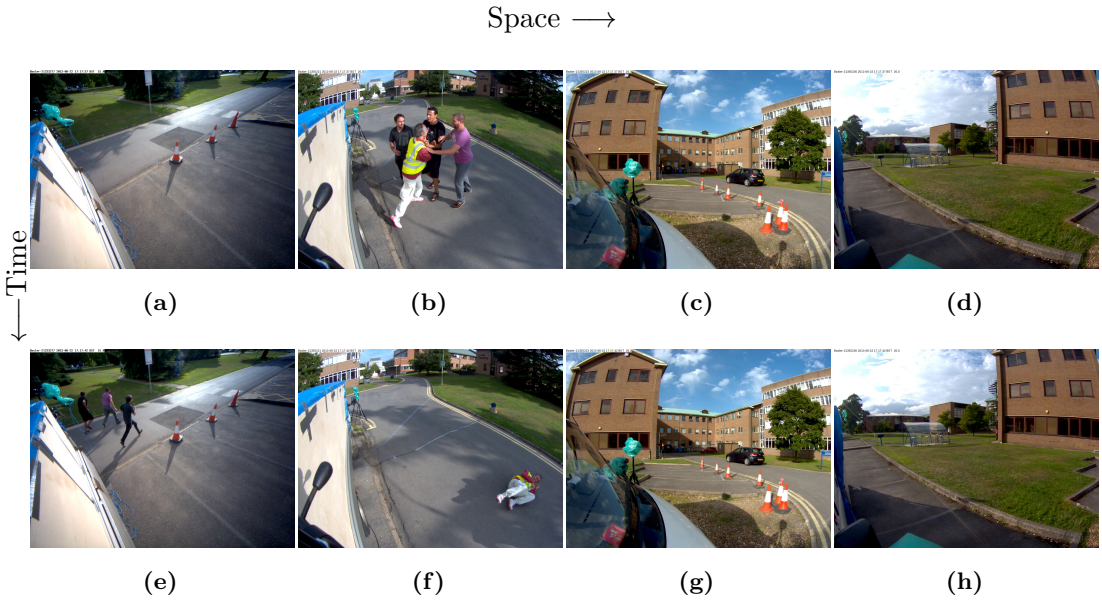


Figure 2.11: Multi-camera surveillance footage from PETS2014 dataset showing a fight incident [35]. Each column shows footage from individual camera and each row shows footage at a different time. (a) (c) (d) (g) (h) Camera views without any foreground activity. (b) Actual fight incident. (e) Perpetrators running away. (f) Driver falling on the ground.

MIPSA in PRISMATICA, which collects image data from all the sensor nodes, verifies the images and sends the results ("yes" or "no") back to the cameras. There are several downsides of this approach:

- The server has to collect and process all the data itself, which may be challenging, particularly in real-time applications.
- Scalability: as the number of cameras increases, it becomes more and more challenging for the server to cope.
- Reliability: if the server is busy or connection to the server fails, the overall system fails to work.
- Security and Privacy: As raw images are transferred to the server, it may be

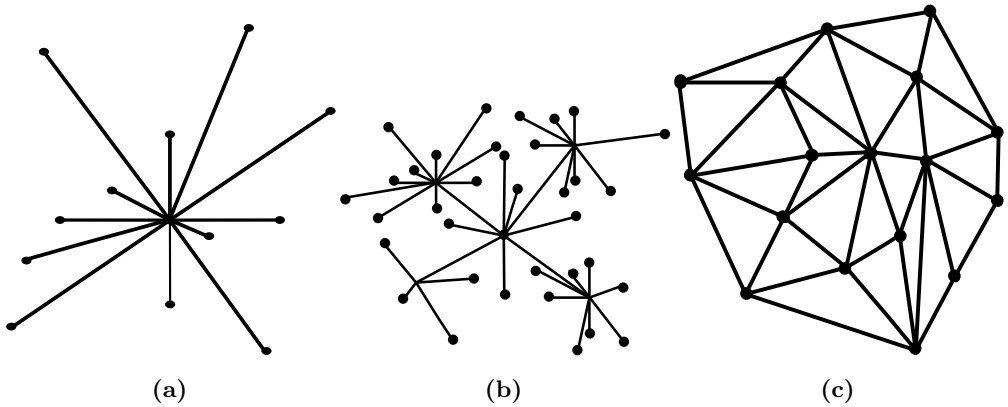


Figure 2.12: Different network topologies (a)Centralised (b)Decentralised (c)Distributed

hacked in during transmission which means the data is no longer reliable. From user's point of view, the central server has access to sensitive information.

2.3.3 Distributed Scenario

The decentralised implementation partially alleviates the shortcomings. The system would be more reliable as there are many mini-servers and the delay would be significantly less. However, the best way would be to implement it as distributed as shown in Figure 2.12. State-of-the-art smart cameras have a significant processing capability as standard. Each sensor can generate a unique encrypted signature for the people in its FOV and maybe also identify them. Since each camera does processing of its own images, they could be powerful just enough to process its own targets only. This could mean less or no communication is required to the central server. Thus, even if some of the connections fail, they could still work fully or partially. If the data is not sent to the server, there is less chance to be hacked in between, so the system is more secure and privacy aware. Furthermore, if commercial off-the-shelf (COTS) devices such as smart-phones are used, the system development cost and time can be minimised. The cameras could be deployed, scaled as well as replaced easily if necessary.

In a military context, distributed implementation means the camera may be embedded

within a soldier's uniform to monitor targets without raising suspicion in conflict zones like the scenario described in Section 1.3. For example, the white mobile cameras in Figure 2.10 could be soldiers observing a suspicious target "2". Whenever required, sensors can communicate a person's unique signature feature to other cameras to establish who "2" is the right target. Such a distributed and decentralised approach would be ideal.

Current Issues

Even though there are numerous benefits, there are problems associated with such a distributed approach. First, the images are captured in two different cameras so transferring images or features of the person will incur a communication cost. The unique person features commonly referred to as "signature" also have to be communicated from one camera to another as shown in Figure 2.9 so that the two signatures can be matched. These are often battery powered devices, such as a smart-phone, so maximising the battery life is desired. As these cameras are connected with wireless networks such as a Wi-Fi or cellular system, the time taken and energy required to send the data across the network is directly proportional to the length of the data [36]. As the signature length increases, the time and energy cost of the signature exchange increases, so the device should keep the signature size as small as possible.

Second, wireless communication still be not secure, making it prone to attack. Even though encryptions are used in the design phase, it cannot be guaranteed during implementation. For example, if the operator or the user at runtime doesn't use any encryption or a weak password, the communication is susceptible to hacking.

Third, there is a computational cost associated with generating the signature used by the PRID system. Distributed systems are equipped with less powerful processors and have fewer resources. As such, the signature generation may be slower compared to a centralised server, and there may be a limitation to the number of persons each camera can identify at a time due to resource limitation on the camera. To overcome these

limitations and execute algorithms with a high time complexity on the resource limited devices, there may be two basic approaches.

- Simplify the algorithm for faster runtime which may sacrifice accuracy.
- Seek alternatives to local execution or perform *Computation Offloading*.

However, for the *Computation Offloading* to be fruitful, the following three essential assessments must be carried out.

1. Choose a suitable PRID algorithm
2. Consider the cost of executing the algorithm on board (local execution).
3. Consider the cost of executing the algorithm on neighbours or the cloud.

2.4 Existing Algorithms

Person re-identification algorithms can broadly be classified into supervised and unsupervised algorithms. Supervised methods include algorithms like the Mid-level features [37], Keep It Simple and Straightforward MEtric (KISSME) [38], Locally Aligned Feature Transform (LAFT) [39], and the Information Theoretic Metric Learning (ITML) [40]. They focus on metric learning [41], whereas unsupervised algorithms focus on feature design. Some popular unsupervised methods include Symmetry-Driven Accumulation of Local Features (SDALF) [42], Bio-inspired Covariance based features (BiCov) [43] and spatio-temporal features [44]. A summary of these algorithms is listed in Table 2.1, for more detailed review of recent approaches, refer to these papers [29, 41, 45–47]. Also there are two general aspects of PRID; identification and verification. Identification means finding the closest match of a probe sample within the given gallery set whereas verification means asserting whether or not two samples are of the same subject (person).

Table 2.1: Comparison of existing hand-crafted PRID algorithms. Deep learning methods are exempted from this as they are all supervised. Key Ref.= Reference, Sup= Supervised

Ref.	Algorithm	Sup	Remarks
[40]	ITML	✓	Mahalanobis Metric Learning
[39]	LAFT	✓	Mahalanobis Metric Learning
[38]	KISSME	✓	Generic feature using Colour and LBP, Mahalanobis Metric Learning
[43]	BiCov		Biologically inspired filters (Gabor filters)
[48]	SDALF		Feature design with Recurrent High-Structured Patches (RHSP) and MSCR.
[44]	Spatio-temporal		Spatial and temporal features made up of Colour histograms and edge information.
[49]	U. Saliency		Features using Colour and SIFT, patch making weighted by saliency of the patch

Among many algorithms, three key ones are selected based on their significance in person re-identification and availability of their source code. To understand their suitability for implementation on an embedded system, these algorithms are discussed here with a focus on their operational steps, complexity and the length of a person's signature feature.

2.4.1 KISS MEtric Learning

KISSME [38] is a supervised method that focusses on learning the metric rather than a complicated descriptor design. The descriptor consists of colour histograms and LBPs [24]. First, the images are divided into overlapping blocks and histograms are extracted in Hue Saturation Value (HSV) and LAB colour-space. Then LBPs are extracted to capture the texture information. For the VIPeR dataset, based on the

code and data[1] provided by the authors [38], each image has 21315 dimension features. Principal Component Analysis (PCA) is used by the authors to shorten the length of the descriptor to 34 experimentally chosen dimensions. For the metric learning, the authors have chosen widely known Mahalanobis Metric learning technique. The Mahalanobis metric, $d^2_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j)$ between two data points $\mathbf{x}_i$ and $\mathbf{x}_j$ is defined as

$$d^2_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j^T)\mathbf{M}(\mathbf{x}_i - \mathbf{x}_j) \tag{2.1}$$

where $\mathbf{M} \succeq 0$ is a positive semi-definite matrix and $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^d$ is a pair of samples (i, j). The main approach of Mahalanobis based algorithms is to define and learn the matrix $\mathbf{M}$ such that distance between images of same class (i.e. intra-class) is minimised and distance between images of different classes (i.e. inter-class) are maximised. KISSME [38], ITML [40], Logistic Discriminant Metric Learning (LDML) [50] and LAFT [39] are based on these methods. A detailed review of Mahalanobis based methods can be found in Roth et al.'s paper [41]. KISSME tries to address the metric learning approach from a statistical inference point of view. They test the hypothesis H_0 that the pair is dissimilar versus the alternative hypothesis H_1 that the pair is similar. This is found by defining the log-likelihood ratio:

$$\delta(\mathbf{x}_{ij}) = \log\left(\frac{p(\mathbf{x}_{ij}|H_0)}{p(\mathbf{x}_{ij}|H_1)}\right) = \log\left(\frac{f(\mathbf{x}_{ij}|\theta_0)}{f(\mathbf{x}_{ij}|\theta_1)}\right) \tag{2.2}$$

where $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$ is the pairwise difference with zero mean, $f(\mathbf{x}_{ij}|\theta_0)$ and $f(\mathbf{x}_{ij}|\theta_1)$ are a probability distribution function (pdf) with parameters θ_0 and θ_1 for hypothesis H_0 and H_1 that a pair (i, j) is dissimilar and similar respectively. A high value of $\delta(\mathbf{x}_{ij})$ means the pair are dissimilar and vice-versa. Assuming a Gaussian structure of the difference space, the authors relax the problem in Equation (2.2) and write the log likelihood ratio as

$$\delta(\mathbf{x}_{ij}) = \log\left(\frac{\frac{1}{\sqrt{2\pi|\Sigma_{y_{ij}=0}|}}\exp(-1/2\,\mathbf{x}_{ij}^T\Sigma^{-1}_{y_{ij}=0}\,\mathbf{x}_{ij})}{\frac{1}{\sqrt{2\pi|\Sigma_{y_{ij}=1}|}}\exp(-1/2\,\mathbf{x}_{ij}^T\Sigma^{-1}_{y_{ij}=1}\mathbf{x}_{ij})}\right) \tag{2.3}$$

[1] accessible from https://lrs.icg.tugraz.at/research/kissme/

where,

$$\Sigma_{y_{ij}=0,1} = \sum_{y_{ij}=0,1} (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T \tag{2.4}$$

Expanding Equation (2.3), it is straightforward to show

$$\delta(\mathbf{x}_{ij}) = \mathbf{x}_{ij}^T \Sigma_{y_{ij}=1}^{-1} \mathbf{x}_{ij} + \log(|\Sigma_{y_{ij}=1}|) - \mathbf{x}_{ij}^T \Sigma_{y_{ij}=0}^{-1} \mathbf{x}_{ij} - \log(|\Sigma_{y_{ij}=0}|) \tag{2.5}$$

Removing the constant log terms, Equation (2.5) further simplifies to

$$\delta(\mathbf{x}_{ij}) = \mathbf{x}_{ij}^T (\Sigma_{y_{ij}=1}^{-1} - \Sigma_{y_{ij}=0}^{-1}) \mathbf{x}_{ij} \tag{2.6}$$

They arrive at the Mahalanobis distance metric in Equation (2.1) that reflects the properties of the log-likelihood ratio test by re-projecting $\hat{\mathbf{M}} = \left(\Sigma_{y_{ij}=1}^{-1} - \Sigma_{y_{ij}=0}^{-1}\right)$. If $\hat{\mathbf{M}}$ is not positive semi definite, Eigen-analysis is performed and any Eigen-value that is negative or zero is changed to a very small positive number, so that $\mathbf{M}$ is a positive semi-definite matrix.

2.4.2 Symmetry-Driven Accumulation of Local Features

Symmetry-Driven Accumulation of Local Features (SDALF) [42] is an unsupervised method suitable for both single-shot and multi-shot images. In multi-shot images, the algorithm may benefit from seeing multiple images. A pedestrian image is divided into the head, torso and leg region and three types of features namely Weighted Color Histograms (WHSV), Maximally Stable Color Region (MSCR) and Recurrent High-Structured Patches (RHSP) are extracted. Each of these features is extracted from the torso and leg region, and optionally from the head region. The histogram feature is built with 12 bins per channel per region, totalling to $12 \times 3 \times 3 = 108$ dimensions[2]. The MSCR feature of a blob regions [26] is represented by 9 dimensional feature, but the number of blobs per image is variable. Similarly, the feature-length of RHSP features is variable as well. The similarity between two images is calculated as the weighted sum of

[2]reduced to 72 if the head region is not used

Euclidean distances between their features. As the algorithm is unsupervised, it does not require any training and is also applicable to video sequences.

2.4.3 Unsupervised Saliency

Zhao et al. defines *Saliency* as "distinct features that 1) are *discriminative* in making a person standing out from their companions, and 2) are *reliable* in finding the same person across different views" [51]. Zhao et al. developed a few variants of supervised and unsupervised methods using saliency [37, 49, 51] but in this work, only Unsupervised Salience Matching [49] is examined.

The key idea here is to use patch matching between the patches of a probe image and the set of images (gallery). However, instead of all matches having same weights, patches that are salient or distinct are weighted more. The algorithm is described in detail here.

First, each image is densely divided into overlapping patches and for each patch, 32 bin histograms are computed in LAB colour-space in three scales. Similarly for SIFT features, each patch is further divided into 4×4 cells to obtain $4 \times 4 \times 8 = 128$ dimensional feature per channel. The combined feature for each patch $\mathbf{x}$ is $32 \times 3 \times 3 + 4 \times 4 \times 8 \times 3 = 672$ dimensions and called "dColorSIFT" . For the m^{th} row of p^{th} image in camera A, the dColorSIFT features is represented as

$$\mathcal{T}^{A,p}(m) = \{\mathbf{x}_{m,n}^{A,p} | n = 1, 2, \ldots, N\} \tag{2.7}$$

where (m, n) denotes the patch centred at the m^{th} row and the n^{th} column of the image. Once the features are extracted for each patch, they need to be matched with the corresponding patch feature of image in a different camera to check whether they belong to the same personnel or not. This is done using a Gaussian based similarity function defined below.

$$g(\mathbf{x}, \mathbf{y}) = \exp(-\frac{\|\mathbf{x} - \mathbf{y}\|_2^2}{2\sigma^2}) \tag{2.8}$$

where $\mathbf{x}, \mathbf{y}$ are the features of two patches and σ is the standard deviation of the Gaussian.

Figure 2.13: Illustration of adjacency constrained search. Green region represents the adjacency constrained search set of the patch in yellow box. The patch in the red box is the target match. In the first example, match is actually found on different shoulder. [51]

However, due to misalignments, patches may appear above or below in the other image. So, a search set called the "Adjacency constrained set"($\hat{S}$) is created containing patches at $\pm l$ rows. The authors selected $l = 2$ in their paper. In Figure 2.13 $\hat{S}$ of the patch in the yellow box is the green region. Mathematically, $\hat{S}$ for $\mathbf{x}_{m,n}^{A,u}$ in image q of Camera B is defined as:

$$\hat{S}(\mathbf{x}_{m,n}^{A,u}, \mathbf{X}^{B,q}) = \{\mathbf{x}_{m,n}^{B,q} | i = \max(0, m-l), \ldots, \min(M, m+l), n = 1, \ldots, N\} \quad (2.9)$$

To calculate the saliency of the patch, the authors used the Nearest Neighbour (NN) distances. For each patch, a NN set is created in the reference set (Camera A) using the equation below.

$$X_{NN}(\mathbf{x}_{m,n}^{A,p}) = \{\mathbf{x} | \underset{\hat{\mathbf{x}} \in \hat{S}_{p,q}}{\operatorname{argmax}} \; g(\mathbf{x}_{m,n}^{A,p}, \hat{\mathbf{x}}), q = 1, 2, \ldots, N_r\} \quad (2.10)$$

where, N_r is the number of images in the reference set. A saliency score is assigned for each patch as

$$s_{knn}(\mathbf{x}_{m,n}^{A,u}) = D_k(X_{NN}(\mathbf{x}_{m,n}^{A,u})) \quad (2.11)$$

where D_k denotes the Euclidean distance to the k^{th} nearest neighbour. A high value of s_{knn} means the patch is different than other patches. A patch which is visually different

than half of the patches in the set were considered to be a salient patch. So, k is set to $N_r/2$. The saliency score in Equation (2.10) is used in unsupervised bi-directional weighted matching to find similarity between two images. This is denoted as Saliency guided Dense Correspondence (SDC) and defined as

$$u(\mathbf{x}^{A,u}, \mathbf{x}^{B,v}) = \sum_{m,n} \frac{s_{knn}(\mathbf{x}^{A,u}_{m,n}) g(\mathbf{x}^{A,u}_{m,n}, \mathbf{x}^{B,v}_{i,j}) s_{knn}(\mathbf{x}^{B,v}_{i,j})}{\alpha + |s_{knn}(\mathbf{x}^{A,u}_{m,n}) - s_{knn}(\mathbf{x}^{B,v}_{i,j})|} \tag{2.12}$$

where $g(\mathbf{x}^{A,u}_{m,n}, \mathbf{x}^{B,v}_{i,j}) = \exp(-\frac{\|\mathbf{x}^{A,u}_{m,n} - \mathbf{x}^{B,v}_{i,j}\|_2^2}{2\sigma^2})$ is the visual similarity between patches defined in Equation (2.8) and α is the penalty of salience difference.

2.4.4 Deep Learning Approaches

This section briefly explores the current state-of-the-art algorithms in PRID. Like most sectors of CV, the latest algorithms are dominated by DNNs. Deep PRID algorithms generally use a bi-forked architecture which uses a pair of images as input similar to the structure shown in Figure 2.9. However, the signature generation is done using convolution neural networks, and there is no communication channel in between matching signatures. This structure is referred to as the Siamese architecture. The earlier approaches such as [52–54] focussed on extracting features from each image from the pair of images using a number of convolution layers and subtracting the resulting features. The difference is then passed to a series of fully-connected layers which then decides whether the pair of the images is of the same person or different. Recently, there have been slight variations in the network architecture, such as the use of triplet loss in [55] and use of an Adversarial network in [56]. The triplet loss network improves the result by regularizing the network using the third image, so that the distance metric learned is not too biased, especially due to limited positive pairs compared to the negative pairs of images in the dataset [55]. This has since been extended to quadruple loss [57]. The problem of limited samples has been addressed by various approaches. The easiest and most effective is the creation of larger dataset. A second approach is

Table 2.2: Rank 1 results of Deep learning approaches on Viewpoint Invariant Pedestrian Recognition (VIPeR) and CUHK01 dataset. For the CUHK01, the number of test subjects are 486

Algorithm	**VIPeR**	**CUHK01**
Deep Metric Learning [52]	28.23	-
Deep ReID [53]	-	27.9
Improved Deep Learning [54]	34.81	47.5
Triplet Loss [55]	47.8	53.7
Quadruplet Loss [57]	48.42	62.55
Deep Transfer Learning [59]	56.3	77

to use a Generative Adversarial Networks (GAN) [58]. The GAN has two networks called Generator and the Adversarial network that compete against each other during the training phase in a zero-sum game [58]. A third approach which is currently the state-of-the-art in PRID uses the transfer learning [59]. Geng et al. argue that the datasets in PRID are tiny compared to others used for training deep neural nets such as Imagenet. So, in their paper, they train their network on Imagenet and transfer the learned weights for the PRID problem. The performance of these approaches far outweighs the three methods described above – see Table 2.2 although the training for these methods use other larger datasets mentioned in Section 2.4.5. For detail reviews on current approaches, interested readers should refer to [60].

Fundamental Blocks of DNN

The fundamental layers of deep PRID algorithms are similar to those used in object classification and face identification. The core layers are the Convolution Layer, the Fully Connected, Pooling and the Loss Layer [61].

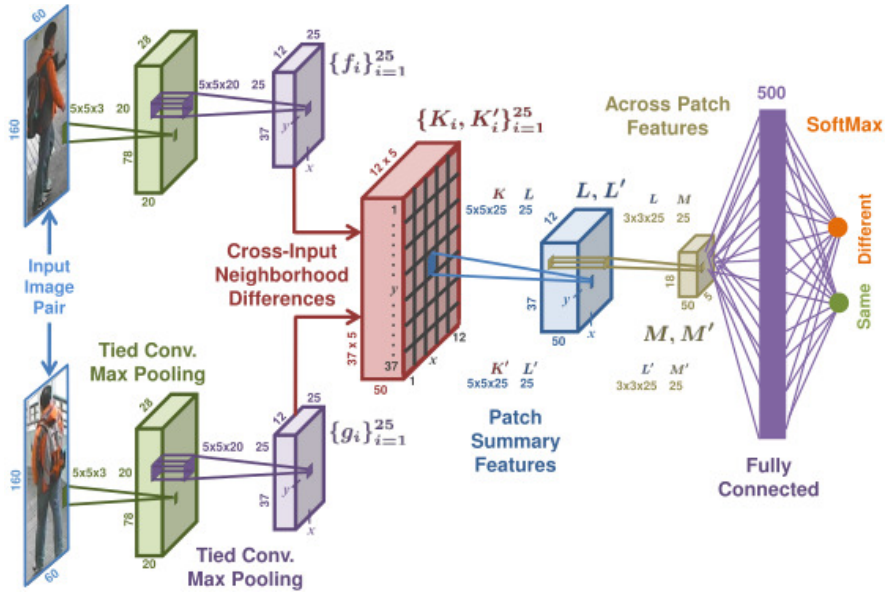


Figure 2.14: General Architecture of a deep Siamese PRID algorithm [54]

- **Fully Connected**: A one layer fully connected network can be expressed as:

$$y_i(\mathbf{x}, \mathbf{w}) = f(\sum_{j=1}^{U} w_{ji} \times x_j + b_i), \quad \forall i \in V \tag{2.13}$$

where, U, V are the number of inputs and outputs respectively; $w_{ji} \in \mathbb{R}^{U \times V}$ and $b_i \in \mathbb{R}^V$ are the weights and biases respectively; and $f(\cdot)$ is an activation function that introduces non-linearity [27]. Common activation functions include the Sigmoid, the Rectified Linear Unit (ReLU) and the leaky ReLU [61]. Generally, the fully connected layer is implemented as a combination of a linear layer followed by an activation layer.

- **Convolution Layer**: A Convolution layer convolves a number of filters called receptor fields around the image. Initial layers of a Convolutional Neural Network (CNN) are good at extracting low level features such as edges and corners whereas higher level layers are generally better at extracting more complex features comprising of many low level features. Compared to a fully connected layer, a convolution layer uses a smaller number of weights, however, the number of computations is significantly higher.

Table 2.3: Operations and memory count for core deep neural layers based on [61]. Filters are assumed to be square shaped (i.e. $F \times F$) and memory calculation is based one symbol takes four bytes

Layer	**Input Size**	**Parameters**	**Output Size**	**Operation#**	**Weights#**	**Memory(B)**
Convolution	$I = W_1 \times H_1 \times D_1$ W_1 = Width H_1 = Height D_1 = Depth	K = No. of Filters F =Filter Size S = Stride P = Padding	$U = W_2 \times H_2 \times D_2$, where $W_2 = (W_1 - F + 2P)/S + 1$ $H_2 = (H_1 - F + 2P)/S + 1$ $D_2 = K$	$(2 \times F^2 \times D_1 - 1) \times U$	$F^2 \times D_1 \times K$	$F^2 \times K \times 4$
Pooling	$I = W_1 \times H_1 \times D_1$ W_1 = Width H_1 = Height D_1 = Depth	F = Filter Size S = Stride	$U = W_2 \times H_2 \times D_2$, where $W_2 = (W_1 - F)/S + 1$ $H_2 = (H_1 - F)/S + 1$ $D_2 = D_1$	$F^2 \times U$	0	$F^2 \times D_1 \times 4$
Linear	I		U	$I \times U + U$	$I \times U$	$U \times 4$

- **Pooling Layer**: A pooling layer subsamples the spatial resolution of the input feature map using average or maximum response. It retains the input depths and does not have any weights attached to it for learning.

Table 2.3 shows the most commonly used layers in DNN and their computational complexity, memory footprint and the number of weight parameters in each layer for a forward pass. The backward pass is about twice as complicated compared to the forward pass. However, an assumption can be made that the DNN is implemented on the embedded device only for inference and not for the training phase. The most computationally expensive part is the convolution layers whereas the fully connected layer contains the most number of weights. The number of operations shown in Table 2.3 is based on a naive implementation. There are many algorithms to calculate convolutions and other related operations more efficiently [62]. For example, Convolution can be performed by multiplication in the frequency domain. However, it is less efficient if the filter sizes are small.

2.4.5 Datasets

Popular publicly available datasets for PRID are listed in Table 2.4 and briefly described below.

VIPeR

VIPeR is the most widely used dataset, one of the reason being the limited number of samples per subject, which makes it one of the toughest dataset in the literature. The VIPeR dataset was used in this work because many published algorithm comparisons are available. This dataset contains 1264 images of 632 people from two cameras. The resolution of the images is just 128×48 pixels and has changing background and variation in pose, which makes it one of the most challenging datasets. Also, the number of images per person per camera is just one, so only single-shot algorithms can run on it.

CAVIAR4REID

The CAVIAR4REID dataset [63] contains snapshots of pedestrians from the CAVIAR video dataset. The challenging aspect of this dataset is that the resolution of the images are varying and often have very poor resolution. The image size varies from 39×19 to 144×72 pixels.

CUHK01

CUHK01 [64] is taken in a campus scenario. It contains images of 971 people from two disjoint camera views and each individual has two samples per camera view. So the total number of images in the dataset is 3884. The image resolution is fixed at 160×60 pixels. They have also released updated datasets CUHK02 [65] and CUHK03 [66].

Table 2.4: Popular Person Re-identification Datasets and their features

Dataset	Unique people	Total Images	Features		
			Pose	Background	Single shot
VIPeR	632	1264	✓	✓	✓
CAVIAR4REID	72	1220	✓	✓	
CUHK01	971	3884	✓	✓	
Market1501	1501	32000	✓	✓	
MARS	1261	1,191,003	✓	✓	

Others

There are several new additions to the PRID datasets. These were necessary as the existing dataset were limited in size and the direction of research has tilted towards deep learning approaches which benefits from large amount of data. For example, Market1501 [67] and Motion Analysis and Re-identification Set (MARS) [68] contains 32,000 and 1,191,003 images of 1501 and 1261 individuals respectively.

2.5 MANET

MANET stands for Mobile Ad-hoc Network and is used to define a type of wireless network that is infrastructure-less and continuously self-configuring [69]. Figure 2.15 shows an example of MANET with two sensor configurations. Each node may be a mobile device such as a smartphones or vehicles with wireless capability, and the edges may be wireless links such as Wi-Fi, Bluetooth, Zigbee. The nodes in the a MANET are mobile and can move in and out of the network randomly. Because of this mobility and the absence of a central scheduler, they may be harder to route to. In an initial arbitrary configuration Figure 2.15a node 2 can connect to node 1 and 5 directly, in the new configuration shown in Figure 2.15b node 2 cannot connect to 1 and connect to 5 via 3 and 7 only.

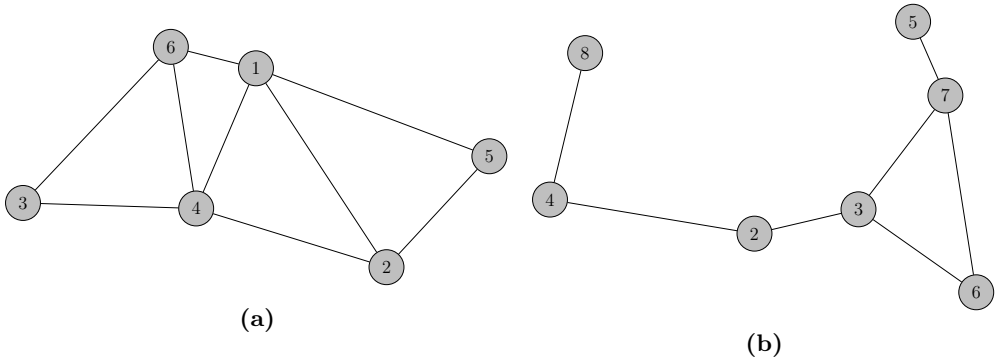


Figure 2.15: An example of a MANET. The circular nodes may be mobile devices such as smartphones, vehicles with connection capabilities. Similarly, the lines between the nodes represent valid connections formed by Wi-Fi, Bluetooth, Zigbee or other device to device communications (a) Initial configuration (b) New configuration after some time. Node 1 has left, while nodes 7 & 8 have joined the scene, also the wireless links are modified based on the new configuration.

2.5.1 Digital Communication System

A digital communication system comprises of a series of complicated processes between the source and the sink – see Figure 2.16. Each line in Figure 2.15 has to go through this process while communicating with each other. An important component in this process is the channel, which refers to the medium of data transfer between the source and the sink.

A MANET uses the same wireless channel as the traditional infrastructure based network and it is shared amongst the nodes, so they are subject to similar fading, noise, and interference [71–73]. They have a significant importance in communication systems as they can limit the rate of data transmission [74, 75] as well as impact the lifetime of a battery powered device [76]. The three main types of propagation effects that can affect wireless transmission are as follows [72]

1. **Deterministic Path Loss**

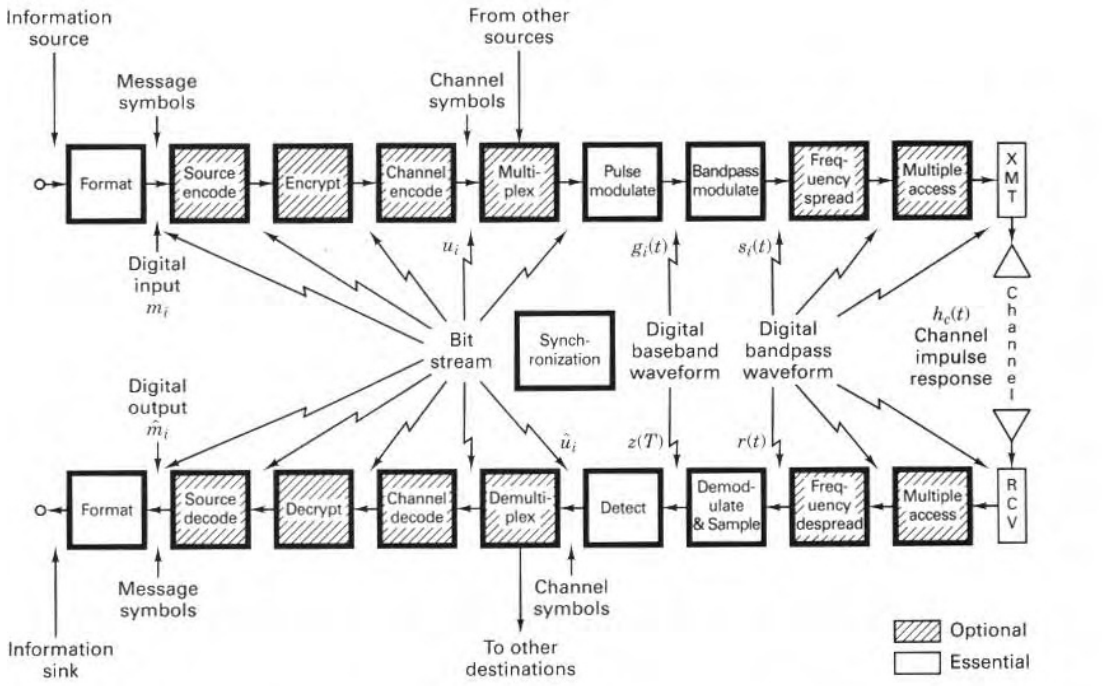


Figure 2.16: Block diagram of a generic digital communications system [70]

This loss is the signal attenuation from the transmitter to the receiver due to the channel medium itself and is proportional to the distance between them. According to [77], it can be characterised as

$$\frac{P_r}{P_t} \propto d^{-\eta} \tag{2.14}$$

where, P_r and P_t represent the received and transmit power levels respectively, d is the distance between transmitter between transmitter and receiver and η is the path loss exponent. For example, in free space the received power can be estimated as:

$$P_r = \frac{P_t G_t G_r \Lambda^2}{(4\pi)^2 d^2 l} \tag{2.15}$$

where, G_t, G_r are the gains of the transmitter and the receiver respectively, Λ is the wavelength of the signal, and l is the system loss factor not related to propagation [78].

2. **Large Scale Fading** This fading is due to the large objects between the source

and receiver. As the result of large scale fading, the average signal strength at the receiver can be higher or lower.

3. **Small Scale Fading**

   Wireless transmission occurs by multiple paths between transmitter and receivers. These paths can combine the wireless signals in constructive and destructive manner changing the received signal strength at the receiver. There are many models used to model these effects such as Rayleigh, Nakagami-m, Ricean distributions etc. [73].

In addition to fading, the signal strength may be deteriorated due to various sources of noise and interference. Interference occurs when two or more nodes in the same area simultaneously try to transmit their data in random access protocols such as Carrier Sense Multiple Access (CSMA). For example, in traditional centralised systems, each node would communicate via the central router or base station. If two devices simultaneously tried to communicate to the server, they would interfere with each other's signal. Random back-off strategies and game theoretic strategies can be used to minimise the interference [78]. The effect of this interference means that the overall data rate is reduced. The data rate of the transmission from a node to the central router can be estimated as [72]

$$R_i = BW \log_2 \left\{ 1 + \frac{P_i H_{i,s}}{\omega + \sum_{k \in S, k \neq i} P_k H_{k,s}} \right\} \tag{2.16}$$

where, Network Bandwidth (BW) is the channel bandwidth, $H_{i,s}$ is the channel gain between mobile device and the base-station, S is the set of sensors, $\omega_n = \omega_i^0 + \omega_i^1$ denotes the background interference power including the noise power ω_i^0 and the interference power ω_i^1 from other nodes. This equation suggests that the rate of transmission can be increased by increasing the power P_i, given that everything else is constant. However, when node i increases its transmission power, the rate for other nodes decreases due to the denominator expression in Equation (2.16). This leads to a game-theoretic scenario whereby nodes compete with each other for the common resources such as channel. Also, there may be difference in MANET interference due to the neighbouring devices. But

in MANET, this may not be the case. For instance, in Figure 2.15b, if nodes 8 and 6 transmits simultaneously to nodes 4 and 7 respectively, and if they are out of their interference zone, they will not interfere with each other's signal.

2.5.2 Routing Protocols

As discussed earlier, routing is challenging task in a MANET [79]. There are numerous variations and types of routing [80]. Two fundamental types "proactive" and "reactive" are presented here. In Chapter 5, when realistic network environments are considered, nodes exchange information in both reactive and proactive arrangements. More details on routing protocols can be found in these surveys [80–82, 82–85].

Proactive (Table-driven)

Proactive routing protocols always maintain a *routing table* that contains up to date information of routes from each node to every other node in the network. This is accomplished by propagating any changes in the network throughout the network. The advantage of this type of routing is that looking for the best route to a node in the network can be carried out instantaneously. However, on the downside, a significant amount of energy can be spent while maintaining the routing tables especially if the nodes are highly dynamic. There are several examples of proactive protocols such as Destination-Sequenced Distance-Vector (DSDV) [86] and Optimised Link State Routing (OLSR) [87]. DSDV is briefly explained here.

In DSDV, each node maintains a *routing table* containing every possible destination in the network along with the number of hops required to reach the destination. The distance is calculated using shortest path algorithms such as Bellman-Ford algorithm [88] and Dijkstra's algorithm [89]. To maintain the *routing table*, two types of updates are propagated in the network. The first is the *full dump* that contains all the table information and can be several packets long. The second is the *incremental* updates which only contain the information about the nodes that have changed. If the network

is stable (ie. nodes only moves infrequently) only *incremental* updates may be necessary and the *full dump* updates are only issued infrequently. However, on the other hand, if the network is dynamic, frequent *full dump* updates, can waste energy as well as bandwidth.

Reactive (On-demand)

Reactive routing works by only communicating and discovering the nodes in the network when required. So it is also known as an "On-demand" or "Source-initiated" algorithm. There are several On-demand routing protocols in the literature, some of the popular ones include Dynamic Source Routing (DSR) [90], Ad-hoc On-demand Distance Vector (AODV) [91] and Temporarily Ordered Routing Algorithm (TORA) [80]. To understand how they work, the DSR algorithm is briefly explained here.

DSR has two phases called *route discovery* and *route maintenance*. When a node wants to send a message, it broadcasts a *route request* packet to its neighbours with the source and the destination id. The neighbours then re-broadcast the packet after adding their own id to it. This process is carried on until it reaches the destination or a node that has a route to the destination their *route cache*. If the (intermediate) node has a valid route to the destination, it adds the route to the messages and sends it back along the path to the sender. When the sender receives the *reply packet* it has a complete route to the destination. The route is maintained by the nodes in their *route cache* based on the *acknowledgements* and *route error*. When there is a problem in transmission, the acknowledgement is not received and the node generates a *route error* packet which is sent back to the sender. The sender on receiving the *route error* message removes the route from its *route cache* and enters the *route discovery* phase. Likewise, AODV is similar to the proactive algorithm DSDV described above except that instead of keeping up to date route information, it creates the routes on-demand.

2.5.3 Mobility Modelling

Capturing the dynamics of how entities move in the physical environment is an integral part of many research areas including MANETs. Collecting real data is not always feasible; also real data may not cover all possible scenarios in the network. So, it is highly desirable to model and simulate real-life movements of users or targets. In this section, Random Walk and the Random Waypoint Model (RWP) is detailed here which is used in Chapters 4 and 5 for modelling target movements in the scene. More details can be found in these surveys and a book [92–94].

Random Walk

The random walk is true to its name; the targets move around in the scene randomly in each step from one point to another. This is similar to the RWP model discussed next but a key difference is that in the random walk, the targets do not pause at any point of the simulation. This is different to how people behave in real-life. For example, pedestrians walking on the road may pause on traffic signals or shop window before continuing their journey again.

Random Waypoint Model

The Random Waypoint Model was introduced by [90] and is similar to the random walk model except for pausing. The birth of the targets follows a Poisson Point Process (PPP). In a PPP, the birth of a target is independent of targets born earlier, and only depends on the mean average rate (λ). The scene is defined as a two dimensional rectangular area with the vertices $\{(x_0, y_0), (x_0, y_1), (x_1, y_0), (x_1, y_1)\}$. The target can be born anywhere in the scene. Once the target is born, it selects its next destination on the scene and the speed of its movement based on two independent uniform random distributions, and move towards it. However, another random variable is used to determine if the target would pause or move. The steps are depicted in Algorithm 1 and an example

of a target moving using RWP is shown in Figure 2.17. The target spawns at location 1 and gradually moves along the path with random speed up to location 9. While the target is in the locations 1 to 9, it may briefly pause before continuing along the path. **Stationary Distribution** The stationary distribution of the target location

Algorithm 1: Pseudocode for RWP

Input: Total simulation time (t_{max}),

Parameter : Boundaries $[\mathbf{x}_{min}, \mathbf{x}_{max}]$, Velocity Range: $[\mathbf{v}_{min}, \mathbf{v}_{max}]$, stopping probability P_{stop}, $\Delta time$ time step

Function : $random(\mathbf{ll}, \mathbf{ul})$ generates a uniformly random number in the range $[\mathbf{ll}, \mathbf{ul}]$

Output: $\mathbf{x}$

Generate $\mathbf{x}_{previous} = random(\mathbf{x}_{min}, \mathbf{x}_{max})$

$\mathbf{x}_{current} = \mathbf{x}_{previous}$, $\mathbf{x}_{next} = \mathbf{x}_{previous}$

while $time < t_{max}$ **do**

  if $random(0,1) > P_{stop}$ *and* $\mathbf{x}_{current} = \mathbf{x}_{next}$ **then**

    $\mathbf{x}_{next} = random(\mathbf{x}_{min}, \mathbf{x}_{max})$

    $\mathbf{v} = random(\mathbf{v}_{min}, \mathbf{v}_{max})$

  end

  Find current position $\mathbf{x}_{current}$ by interpolation using $\mathbf{x}_{previous}, \mathbf{x}_{next}, \mathbf{v}$ and $time$

  $time = time + \Delta time$

end

in RWP should be uniform across all the area of the scene [95, 96]. However, it has been shown that in a long run, the targets concentrate more in the middle of the rectangular scene than towards the edges. So, the stationary probability distribution of the target position is more like a Gaussian than a flat uniform distribution – see Figure 2.18. This is actually beneficial for the simulation in this work. The non-uniform distribution of targets in the scene means sensors at different spatial locations see different target density. Sensors situated on the centre of the scene would detect more targets than the sensors that are situated on the edges.

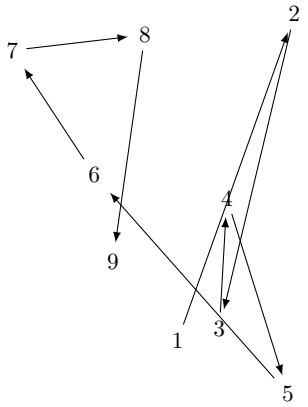


Figure 2.17: Track of a target based on RWP for nine steps. The target starts at location 1 continues along the path with different speed until it stops at 9. During the movement, the target can pause at any step.

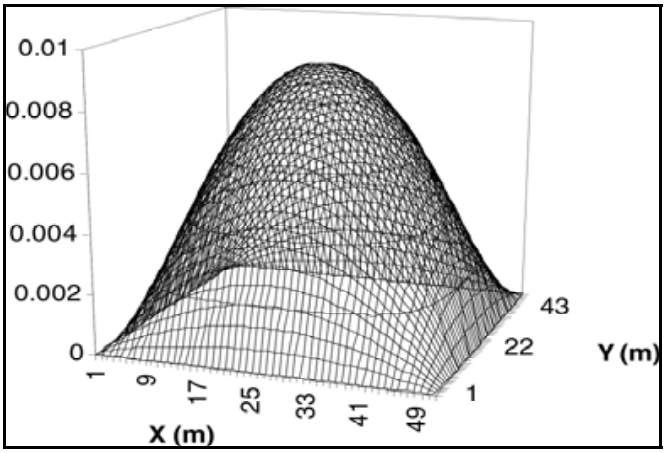


Figure 2.18: Stationary probability distribution of the target posion in a rectangular scene [95]

Other Models

In addition to the Random walk and the RWP models, there are several other mobility models that are used in the literature such as Gauss-Markov model, Pathway Mobility model and Reference Point Group model. [92].

2.6 Queuing Theory

A service provider may not be able to serve all those waiting to be served at once, as it is busy serving others, this gives rise to queueing. For example, in day to day life, people have to line up in front of shopping checkout to be serviced or wait outside the cinema to go inside. Queueing theory studies these type of situations in order to maximise the use of limited resources as well as minimise the delay faced by the customer. It is an important topic useful in many areas including Operational research, Telecommunication, and Computer Science [97, 98]. Queuing theory offers various performance measures that are vital to analyse how well the system is performing and what could be done to make it better. For example, imagine a computer servicing customers. As the rate of arrival of the customers goes up, the average time each customer has to wait becomes longer and eventually it will exceed the acceptable time limit. The question is, should the existing computer be replaced with a faster computer that can service the customers faster or should the number of service stations be increased. Queuing theory can help answer these types of question. In the following section, one of the most fundamental queue type is introduced which will be used in Chapter 5.

2.6.1 Elementary Queue

A queue is generally described using Kendall's notation [97] in the following way

$$A/B/m - \text{queueing discipline} \tag{2.17}$$

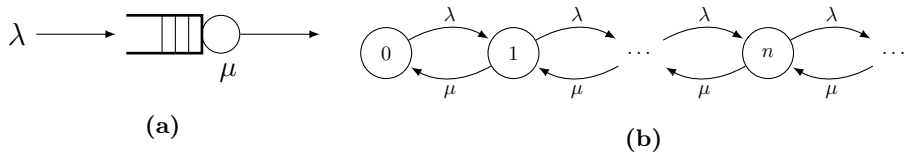


Figure 2.19: An elementary queue. (a)A $M/M/1$ queue with arrival rate of λ and service rate of μ. (b) Markov chain representation of the $M/M/1$ queue. The states represent the number of items in the queue.

where A indicates the distribution of the interarrival times, B denotes the distribution of the service times, and m is the number of servers ($m \geq 1$). Among many possible alternatives, the most common distribution used for A and B is M for Exponential (M stands for Memoryless or Markovian). Similarly $m = 1$ means there is only one server and the queueing discipline is First Come First Service (FCFS). A $M/M/1$ queue, depicted in Figure 2.19a has Poisson Arrival and its service time is exponentially distributed and comprised of one server operating in FCFS basis. An important assumption of $M/M/1$ is that there is infinite buffer so the queue can grow to infinity. While this is not true for physical systems, it can make the modelling easier than the finite buffer systems such as $M/M/1/K$ which has a finite buffer of K spaces. The mean arrival rate of the Poisson arrival is denoted by λ and the service rate is denoted by μ.

A $M/M/1$ queue can be represented by Markov Chain (MC) as shown in Figure 2.19b. The states of the MC is denotes the number of items in the queue. It can be used to find the transient and the steady state solution of a queue. Also, the performance measures that are of interest are many such as:

- Number of items in the queue.
- Utilisation of the server.
- Average waiting time for each item.

Utilisation

Utilisation is defined as the fraction of time in which the server is busy or occupied. In $M/M/1$ queue, there is no limit on the number of jobs in the queue and the server utilisation is given by

$$\rho = \frac{\text{arrival rate}}{\text{service rate}} = \frac{\lambda}{\mu}. \tag{2.18}$$

A queue is defined as rate stable if $\rho < 1$.

Number of Jobs

Let p_k be the probability that there are k jobs in the queue. Then, average number of jobs in the queue can be estimated as

$$L = \sum_{k=0}^{\infty} k p_k. \tag{2.19}$$

Little's Law [97] states that the number of jobs in the queue is equal to the arriving rate times the *response time* which is the time spent in the queue and the server. So, the average number of jobs can be calculated as

$$L = \lambda W \tag{2.20}$$

where, W is the response time.

Throughput

Throughput is the average number of jobs completed by the server in a single unit of time. When, the utilisation of the queue (ρ) is less than one, the throughput is equal to the arriving rate (λ)

2.6.2 Network of Queues

More often in a system, there are multiple service stations or resources that the customers can queue in front of, and they can move to another queue after being serviced by one queue. For example, in real life, during a hospital visit, a patient first of all waits for their turn to see the doctor. The doctor may ask the patient for a blood test or "X-ray" image depending on the case. The patient then waits for the blood test facility or the imaging facility for their turn. Upon completion of these task they have to queue again to see the doctor with the report and so on. These system can be modelled using network of queues. There are two fundamental types of networks of queues; *open* and *closed*. In a closed system, all the arrivals in all the queues are from within the system and not from outside the system. Whereas, in the *open* system, at least one of the queue should accept arrivals from outside the system. In this section, the *open* network of queues is briefly discussed.

Open Network

The Open network is the one in which the system is open to new customer, so at least one of the queues accepts new jobs. For example, the system in Figure 2.20 is a open network because there are external arrivals γ_1, γ_2 coming into the system. Jackson's work builds on the Burke's theorem [97] and states that the queues even in the presence of feedback loops behave as if they were fed by Poisson arrivals, when in fact they are not. A *Jackson network* is an open network of queues with following assumptions [97]

- All the jobs belong to the same class and service times are exponentially distributed.
- The queue length is not finite.
- The service discipline is FCFS.
- Each queue can have external arrivals that are Poisson and a job can leave the network from any node.

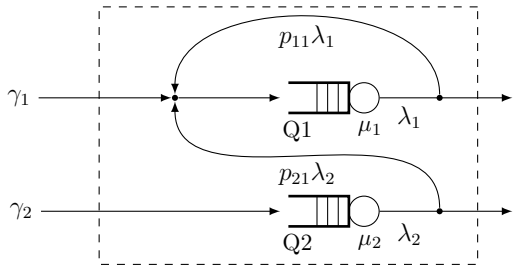


Figure 2.20: A network of two Queues. Total incoming target rate at Q1 (λ_1) is the sum of external target rate (γ_1) and targets rates emanating from the queues heading to Q1. Under stable condition, outgoing rate is equal to the incoming rate.

- After completing service at node i, a job will proceed to node j with probability p_{ij} which is independent of the previous history or will depart from the system never to return again with probability $1 - \sum_{j \in Q} p_{ij}$ where Q is the number of queues in the system.

Following these assumptions, the arrival rates of all queues can be calculated as:

$$\lambda_i = \gamma_i + \sum_{j \in Q} p_{ji}\lambda_j \tag{2.21}$$

where, γ_i is the rate of arrival of external targets at queue i, λ_j is the arrival rate at queue j, p_{ji} is the probability a job moves from queue j to i. The arrival rates of the two queues $Q1, Q2$ in Figure 2.20 is calculated as

$$\begin{aligned} \lambda_1 &= \gamma_1 + p_{11}\lambda_1 + p21\lambda_2 \\ \lambda_2 &= \gamma_2 + p_{12}\lambda_1 + p_{22}\lambda_2. \end{aligned} \tag{2.22}$$

The Jackson network will be used in Chapter 5 to model sensor behaviour.

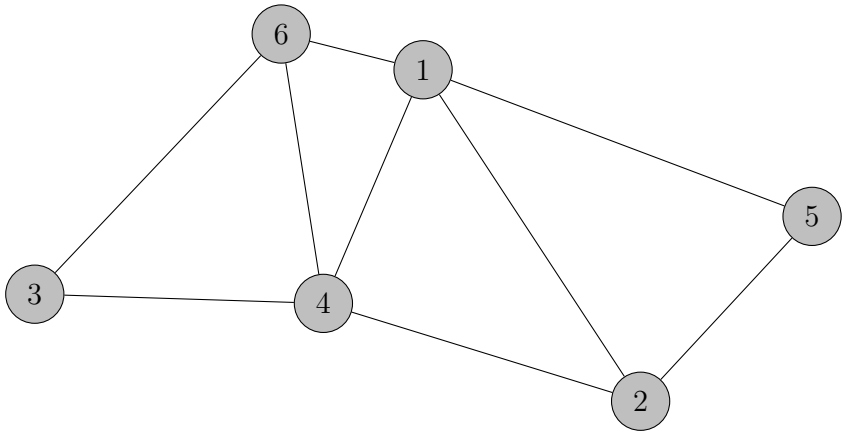


Figure 2.21: MANET as a graph; the sensors are the elements of set V and paths are elements of set E

2.7 Graphs and Network Flows

The MANET described in the chapter can also be represented as graphs and the subsequent model of network of queues can be formulated as a network flow problem. This idea is used in Chapter 5, hence it is described very briefly here.

2.7.1 Graphs

According to Diestel, "*A graph is a pair* $G = (V, E)$ *of sets such that* $E \subseteq |V|^2$*; thus the elements of* E *are two element subsets of* V" [99]. The elements of the set V are called 'vertices' or 'nodes' and the elements of E are called 'edges' or 'arcs'. The MANET examples shown in Figure 2.15 is repeated in Figure 2.21; the nodes belong to the set of vertices ie. $V = \{1, 2, \ldots, 6\}$ and the paths belong to edges set ie. $E = \{\{1,2\},\{1,4\},\{1,5\},\{1,6\},\{4,2\},\{4,6\},\{3,4\},\{3,6\},\{5,2\}\}$. The nodes that share an edge are called adjacent nodes. In this example, 3, 6 are adjacent nodes whereas 1,3 are not. The connectivity of the graph G can be described using an adjacency matrix (A). A_{ij} is set to 1 if vertex i can communicate with vertex j, and vice versa. The

adjacency matrix for the network in Figure 2.21 can be written as

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{2.23}$$

It can be noticed that A in Equation (2.23) is symmetric along the main diagonal (ie. A is equal to its transpose A^T). This is because, the graph in consideration is an undirected graph. In the case of directed graph, A_{ij} may not be equal to A_{ji}. The adjacency matrix is used in Chapters 4 and 5 for conducting searches for neighbours of sensors and maintaining active neighbours.

2.7.2 Network Flow Problems

Graphs and networks can be used to model and solve many real life problems. For example, finding the best path between two nodes can be related to navigation problem for an autonomous robot or drone; similarly, maximizing the traffic and utilizing the resources in the best possible manner can be accomplished using network flows. Ahuja et al. defines three basic network flow problems [89]

Shortest Path Problem

The shortest path problem is to find the cheapest way to get from one point to another. The routing protocols defined in Section 2.5.2 such as DSDV and AODV use these algorithm to calculate the cheapest path from one node to another. In Chapter 5, partial connectivity between nodes is explored and the cost is calculated based on shortest path algorithm.

Maximum Flow Problem

The maximum flow problem is related to maximising the use of resources. The edges or the communication links do not entail cost but rather has capacity limits. So the problem is to find the best path to route from the source node to the destination node with maximum capacity.

Minimum Cost Flow Problem

The minimum cost flow problem is more general network flow problem. The flows in the arcs of the network has cost per unit flow and the problem is to route from source node to destination node with the minimum cost and subject to constraints such as capacity on each arc. In Chapter 5, the network of queues model is solved using minimum cost flow approach. The problem is

$$\text{minimise} \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{2.24a}$$

$$\text{subject to} \tag{2.24b}$$

$$\sum_{\{j:(i,j)\in E\}} x_{ij} - \sum_{\{j:(j,i)\in E\}} x_{ji} = b(i) \qquad \forall i \in V \tag{2.24c}$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \qquad \forall (i,j) \in E, \tag{2.24d}$$

where, c_{ij}, x_{ij} are the cost per unit flow and the flow in the arc (i,j), $b(i)$ is the demand in node i, l_{ij} and u_{ij} are the lower limit and the upper limit of the flow in arc (i,j) respectively. In other words, the problem is to full fill the demand $b(i)$ in all nodes of the network at the minimum cost and subject to lower and upper limit of all arcs. In this particular example, the optimisation problem is linear and can be solved using linear solvers such as simplex and interior point algorithms [89].

2.8 Summary

This chapter covered the fundamental concepts and terminologies that are used throughout the thesis. A comprehensive literature on existing PRID is also presented. The next chapter explores the possibility of *Computation Offloading* and the factors that are involved.

Chapter 3

Motivation for Offloading

3.1 Introduction

The primary objective of this chapter is to evaluate if *Computation Offloading* may be beneficial in application requiring computationally intensive algorithms and the factors that need to be considered. In order to evaluate, three main experiments are performed. The first experiment is to assess which existing algorithms would be the most suitable based on time complexity and the amount of data communication required. The second experiment is to measure the time and energy of executing complex algorithm on a resource limited device such as a smartphone. The third experiment is to quantify the cost of communication based on channel condition as well as device settings. These experiments play an important role in designing *Computation Offloading* algorithms in Chapter 4 and Chapter 5

3.2 Relative Complexity of Person Re-identification (PRID) Algorithms

The first experiment compares existing PRID algorithms. The three algorithms that are compared are Keep It Simple and Straightforward MEtric (KISSME), SDALF and Unsupervised Saliency which were described in Section 2.3. These algorithms were chosen for their importance in the history of PRID and the availability of their source codes.

3.2.1 Algorithmic Performance

The algorithmic performance of the algorithms using Cumulative Matching Characteristics (CMC) [100], which is widely used in person re-identification performance evaluation. It treats PRID as a ranking problem. Rank-1 implies that the correct match has been found whereas Rank-k implies there are $k-1$ wrong classes ahead of the correct class. Cumulative Matching Characteristics (CMC)(k) measures the probability that the correct match has a rank equal or higher than k [45].

The performance of the three algorithms are shown using a CMC graph in Figure 3.1. Unsupervised Saliency [49] has the best Rank 1 results among three at 27%. So on average 27% of the identities are correctly identified in the trials. This is comfortably better than the other two by approximately 7%. SDALF is the worst based on performance alone, which can be visualised in Figure 3.1 by the lowest slope among three. KISSME on the other hand is in between. Actually, at around 6^{th} rank it overtakes the Unsupervised Saliency as well. Based on the performance only, it is easy to choose Unsupervised Saliency as the best algorithm, so Next section looks at the complexity. In an ideal scenario, the algorithms would be implemented on a real distributed system such as an Android smart phone and the results could be measured. However, as the existing algorithms are written in MATLAB, the simulations are carried out on MATLAB running on a desktop PC. Instead, an application (commonly referred to as *app*) is

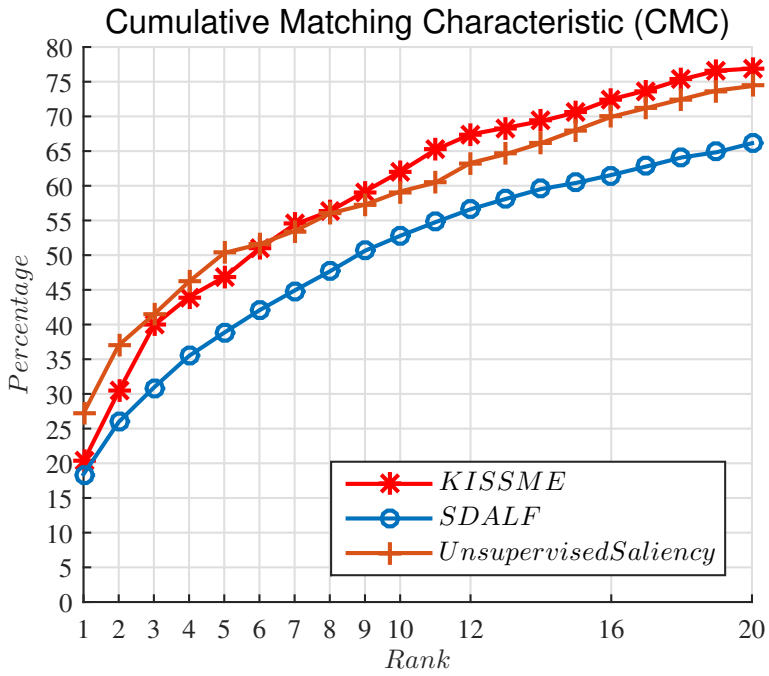


Figure 3.1: Performance of the algorithms in VIPeR dataset

developed to run a Deep Neural Network (DNN) on an Android phone (–see Section 3.3). Experiments are carried out on a desktop PC with an Intel Xeon processor (X5650) with 12 cores and 24 gigabytes of Random Access Memory (RAM) running Scientific Linux 6.5 unless specified. Each algorithm is run for ten times in a Monte Carlo setting. Some of the algorithms have parallel implementation as well but it has been turned it off for these experiments for the following two reasons.

1. Each parallel MATLAB instances run within their own Java Virtual Machine (JVM) environments accounting for increased memory allocations. This caused some algorithms to fill the RAM to the maximum and slowing down the execution.

2. To make the comparisons fair as other algorithms may benefit from parallel implementations as well.

The experiments are conducted for ten Monte Carlo runs. For each run, the Viewpoint Invariant Pedestrian Recognition (VIPeR) dataset is randomly split into two sets of 316

Table 3.1: Average feature length, runtime and Rank 1 results for the compared algorithms in 10 Monte Carlo runs.

Algorithm	**Feature Length**	**Feature Length after Principal Component Analysis (PCA)**	**Time (sec)**	**Rank 1**
KISSME	21315	34	260.4	18.03
SDALF	5359	-	11981.17	19.80
Unsupervised Saliency	201600	-	3453.1	27.22

image pairs each. One set is used for training (for the supervised KISSME) and other for testing. This is similar to the test conventions in these papers [38, 42, 49].

3.2.2 KISSME

Among all the methods, KISSME is the fastest to run. The code available from their website had the features already extracted. However, to calculate the time taken for feature extraction, the code is written as described in their paper [38]. First the image is divided into overlapping blocks of size 8×16 and with stride of 8×8 to obtain 105 patches. Histograms of 24 bins per channel and a uniform Local Binary Pattern (LBP) of 59 bins are computed for each patch. So in total, the feature size is $105 \times 3 \times 2 \times 24 + 105 \times 59 = 21315$ dimensions. The histogram extraction of Hue Saturation Value (HSV) and LAB and LBP features took approximately 260 seconds. They used PCA to reduce the feature dimensions. The training is orders of magnitude faster and of around 0.05 seconds. Nonetheless, feature extraction per image would take about $260/1264 \approx 0.2$ seconds. After dimensionality reduction, the feature dimension is reduced to 34, which is very lower compared to others.

Table 3.2: Detail Execution times for SDALF along with the inference time

Step	Total time(sec)	Inference time (sec)
Division into 3 parts	162.15	0.13
MSCR Extraction	138.21	0.11
WHSV Extraction	123.17	0.10
RHSP Extraction	4824.6	3.81
MSCR Matching	6095.3	9.64
WHSV Matching	214.74	0.34
RHSP Matching	423.00	0.67
Total	11981.17	14.81

3.2.3 SDALF

As discussed in Section 2.4.2, the feature length of Symmetry-Driven Accumulation of Local Features (SDALF) is not fixed but dependent on the number of Recurrent High-Structured Patches (RHSP) patches and Maximally Stable Color Region (MSCR) regions found in the image. Table 3.2 shows the breakdown of average time spent per step for the VIPeR dataset. Based on the number of images in the dataset, inference time on the device is estimated. As seen in Table 3.2, most of the time is spent on MSCR matching and RHSP feature extraction. Experiments are carried out without using RHSP features. Figure 3.2 shows the performance of SDALF on a CMC graph with and without RHSP features. The result showed there is only marginal degradation of performance. The test is conducted in VIPeR dataset.

3.2.4 Unsupervised Saliency

Saliency learning has the highest feature size per image of 201600 dimensions which is almost ten times the length of KISSME features before PCA and 40 times that of SDALF. If it is of MATLAB double precision, it's size is approximately 1.5 Megabytes. Each probe patch has it own adjacency search area for each image in the gallery set.

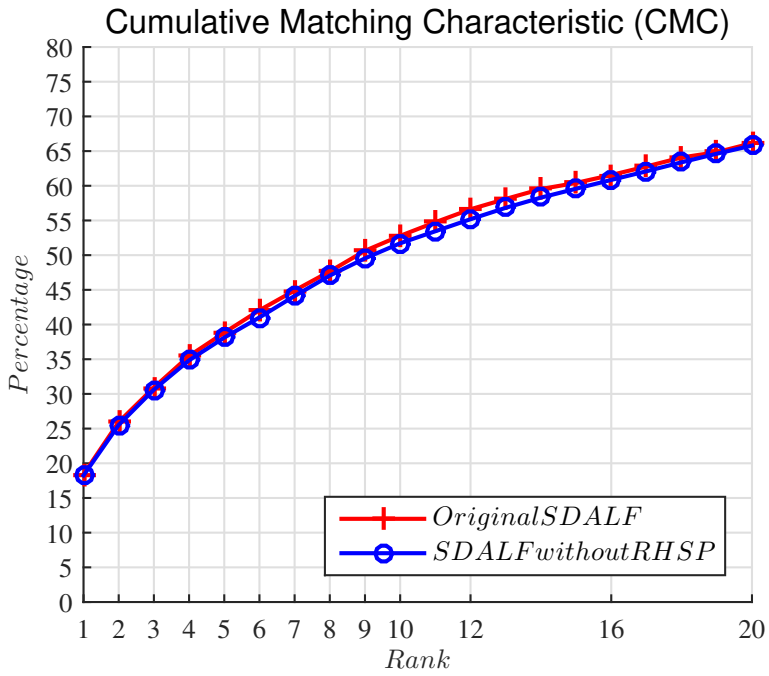


Figure 3.2: Performance of SDALF with and without RHSP

If there are 10 patches per row and constrained search area to be ±2 rows, and there are 100 images in the gallery then. For each patch, the distance between itself and $100 \times 10 \times 5 = 5000$ patches needs to be calculated[1]. If there are 300 patches per image, it amounts to $5000 \times 300 = 1,500,000$ distances per image, which is more than 11 Megabytes in MATLAB double precision. In terms of running on embedded devices, memory is often a limited resource. Similarly, the time complexity of the algorithm is high. The algorithm to calculate the saliency of a patch is at least $O(N^5)$ in the worst case and the patch matching is also in the order of $O(N^3)$. This is reflected in the time taken to run the algorithm – see Table 3.1.

3.2.5 Summary

Saliency has better performance although it is computationally expensive and high data size. SDALF has the lowest feature size before dimension reduction but took the most

[1] except for two top and two bottom rows

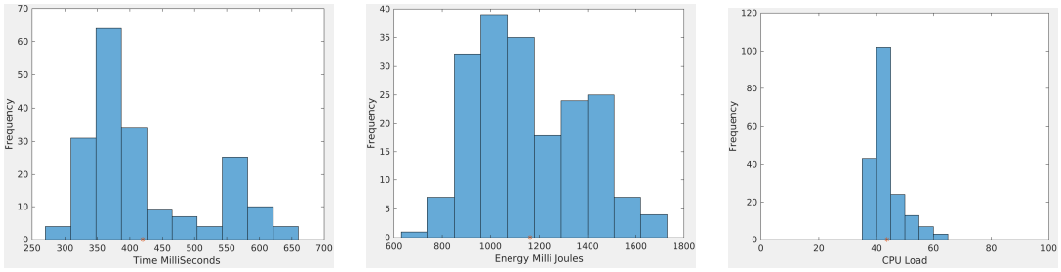
time to execute. KISSME on the other hand, is the best compromise to implement on the distributed systems as it is shown to be fastest to compute and has lowest feature size after PCA.

3.3 Running DNN on a Smartphone

The previous experiment compared three existing algorithms in terms of their algorithmic performance as well as their algorithmic complexity and data footprint. However, they were conducted on a desktop machine with significant amount of computing and memory resources. In order to establish the consequences of executing computational algorithms on embedded devices in terms of resource utilisation, this experiment ran DNN on smartphone device. A smartphone is chosen as the hardware and software of commercial off-the-shelf (COTS) smartphone is relatively powerful as well as inexpensive. However, a COTS smartphone can have many background processes running on them which can make them unpredictable in terms of service guarantee. Instead of trying to eliminate these background processes, this work includes them as part of the model. To run algorithms on a smartphone, a mobile *app* is created and in order to profile the *app* in terms of time and energy, Trepn is used. Trepn and how it is used to profile the *app* is briefly described in Section 3.3.1

A DNN based classification application based on Googlenet [101] and implemented on Tensorflow [102] is tested. The resource usage of each classification operation can be seen in Figure 3.3. It shows significant amount of Central Processing Unit (CPU) usage along with time and energy Figures 3.3a and 3.3b. Based on this experiment, the average energy cost of processing one frame is about 1.2 Joule. If the frame rate of the camera is a nominal 5 Frames Per Second (FPS), the processing cost is 6 Watt. Even if the energy cost of the image sensor is ignored and thermal effects due to continuous running of the CPU are neglected, a fully charged battery of $12Wh$ will run out completely in merely two hours.

An interesting observation is that, for each frame processing it utilised around 50% the



(a) **(b)** **(c)**

Figure 3.3: Resource Utilisation for executing Classification Algorithm based on Googlenet on Sony Z5. (a) Time Usage (b) Energy Usage (c) CPU Usage

CPU on average – see Figure 3.3c. Even though the application that is executed is a DNN classification, it is of very similar architecture to deep PRID algorithms consisting of convolution layers and fully connected layers. PRID algorithms scale with number of person in the scene. If there are more than two people in the scene on average, the system would be overrun and struggle to cope with the computation demand.

3.3.1 Trepn Profiler

Trepn Profiler is developed by Qualcomm to collect vital information from the Android devices such as CPU usage, CPU frequency and battery level. For the Snapdragon processors, they have access to hardware counters in the processor which are not publicly available. It isolates the energy used by an application, by collecting baseline energy consumption before starting the test *app*. At the start and the end of the communication event, the *app* creates an Android Intent recognised by Trepn and broadcasts it. Trepn listens for those Intents and can attribute the energy usage to each event, which is written to a file after the experiment. The timeline is shown in Figure 3.4. It can log several device information such as multi-core CPU load, CPU frequencies, Graphical Processing Unit (GPU) load and Radio States. In some cases, the data reported is not plausible. For example in the tests performed, cellular radio states and GPU load were always reported as zero on the Sony Z5 phone and energy usage was zero at all times for the Samsung Galaxy S4 phone, even though both models uses the Qualcomm's

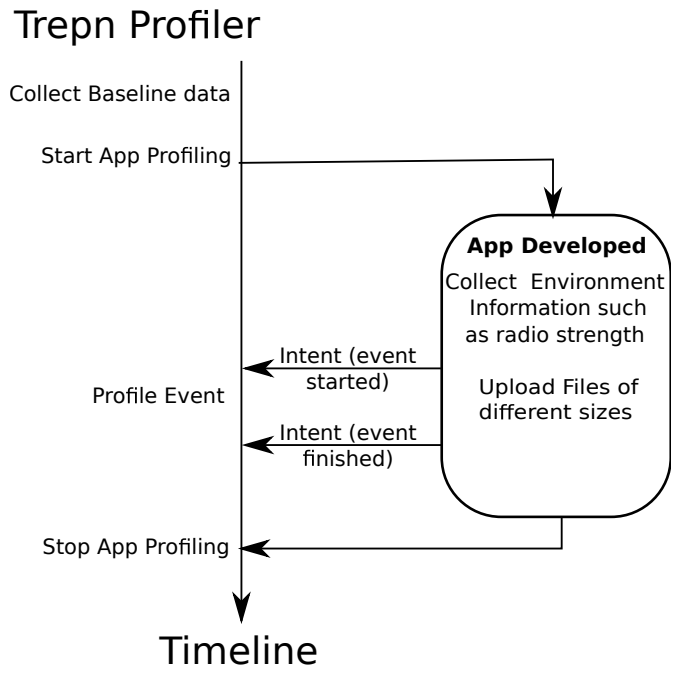


Figure 3.4: Estimating time and energy cost of communication

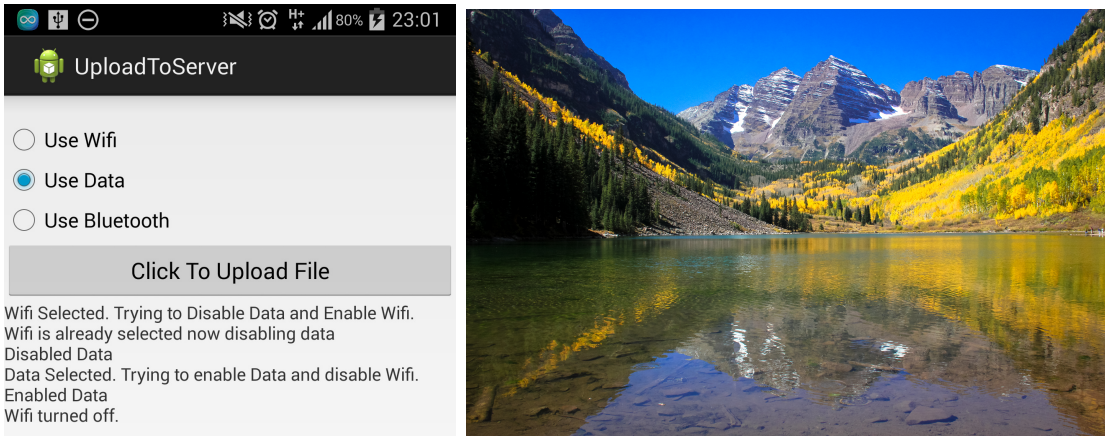
Snapdragon processors. However, as it always report zero when the information is faulty, it is easy to understand that the information is not correct.

3.4 Communication Cost

In the distributed case, the signature of a person extracted in one camera has to be transmitted to another via a communication channel as shown in Figure 2.9. Similarly, if the Smartphone decides to offload the computation to the cloud rather than executing itself it has to send the data to the cloud. The implication of transferring data to a neighbour node or the cloud has a cost in terms of energy and time, particularly in the case of wireless transmission. The upload and download link speed are asymmetrical for both cellular and WiFi, with the download speed generally being faster. In order to find the worst case, a simple Monte-Carlo experiment is setup to upload data using various available communication modalities. An application (commonly known as an *app*) is developed for the Android platform to conduct the experiments and send files of various but known sizes to the server using WiFi or the mobile data (see Figure 3.5a).

Table 3.3: Android Devices used for the experiments

Model	Modality	Description
Sony Z5	Cellular and WiFi	Features Qualcomm MSM8994 Snapdragon 810 chipset with Octa-core Processor and 3 GB RAM
Amazon Fire 8	WiFi	Features Mediatek MT8163 chipset with Quad-core processor and 1.5 GB RAM



(a) (b)

Figure 3.5: An experiment to estimate communication costs. (a) Android application for calculating time and energy cost of transmitting data. (b) Original test image with resolution 5184×3456 and size 13.2 MegaBytes. The image is compressed at several JPEG compression ratio to get test files of different sizes

The application is built using Google's Android Development Kit (ADK) and tested on the two devices shown in Table 3.3. A JPEG image shown in Figure 3.5b is selected as the data to be sent. Various file sizes are generated. The experiments are carried out under various environments such as at different times of the day, indoor and outdoor scenarios.

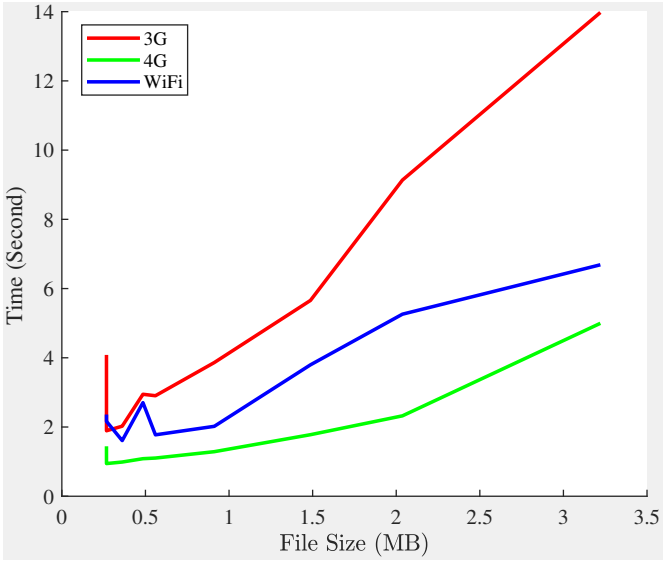


Figure 3.6: Average time cost of uploading files of different sizes.

Time Taken

Time taken is measured using the system clock by subtracting the initial time from the final time, after an acknowledgement is received from the server. The time measurement is accurate to milli-seconds. Figure 3.6 shows average time taken to upload the files. As expected, the time required to upload a file is directly proportional to the file size. It took slightly more than second to transmit 1 MB over 4G, about 2 seconds by WiFi and upto 4 seconds by 3G. 4G is consistently the fastest amongst the three modalities and 3G could be approximately two times slower than the WiFi and three times slower than the 4G. However, the results had significant variances. The standard deviations for the results were about 3.5, 0.5 and 4 seconds for 3G, 4G and WiFi respectively.

The wireless channel and the environment can play a big role on its performance, so the results are further analysed based on the Wireless signal strength. Figure 3.7 shows the actual datapoints along with a polynomial fit on the datapoints. The linear fits shows that file size is bigger factor for time cost than the signal strength although at larger data sizes, the effect of signal strength becomes more visible. As the signal strength is weakened, the time taken to send data grows. Also, the 3G results has more uncertainty than the 4G, there could be two reasons behind this. First, 3G takes longer,

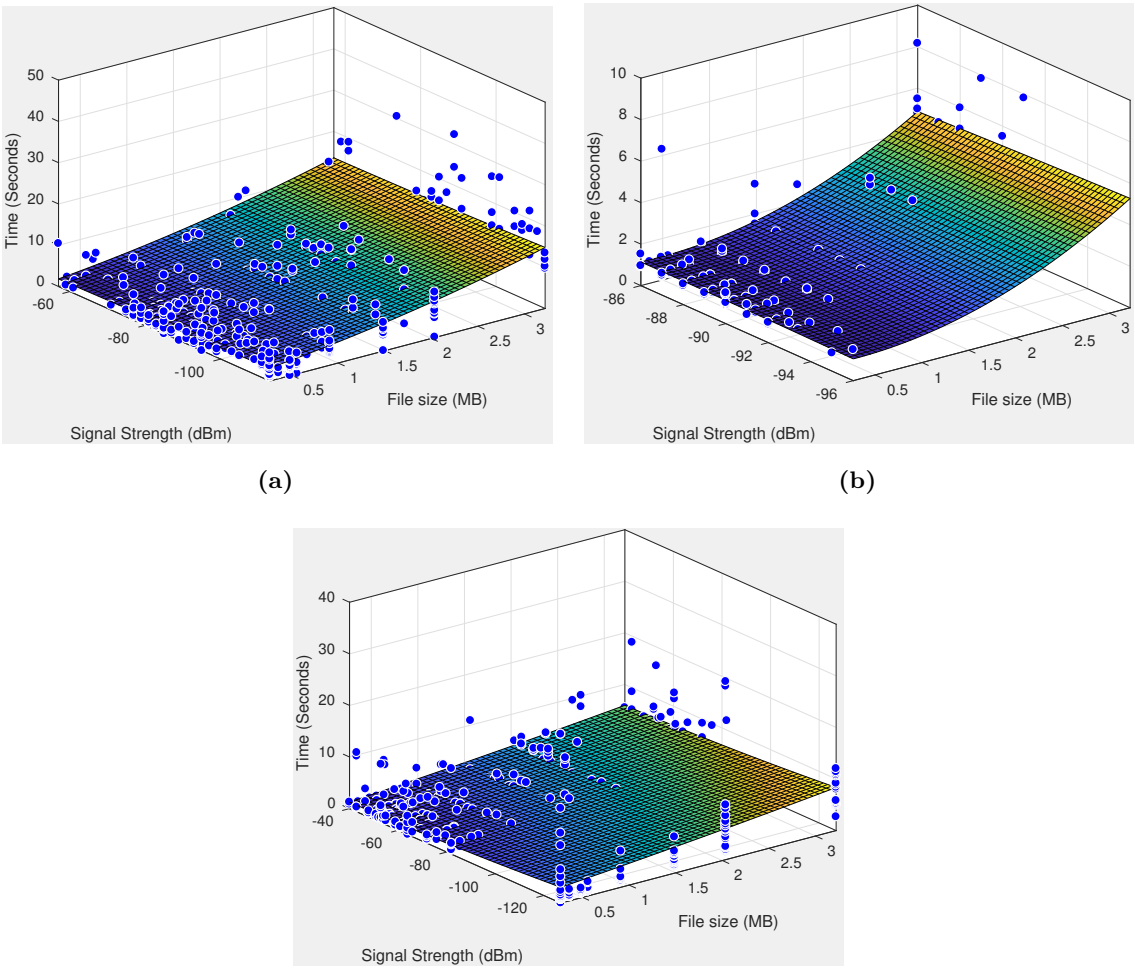


Figure 3.7: Time required to upload files of different sizes under different communication modalities and signal strengths. (a) 3G (b) 4G (c) WiFi

so the variation is more visible. Second, there are numerous wireless technologies termed as 3G with large variations in speed. For example, High-Speed Uplink Packet Access (HSUPA) has maximum theoretical uplink speed of 11.52 Mbps (16QAM) [103] and Wideband-Code Division Multiple Access (W-CDMA) had initial maximum uplink speed of 0.3 Mbps [104], even though both are termed as 3G. It is not possible to experiment with the full range of Received Signal Strength Indication (RSSI) for the cellular because as the signal strength is weakened, the device automatically switched back to previous generation technology for example, from 4G to 3G.

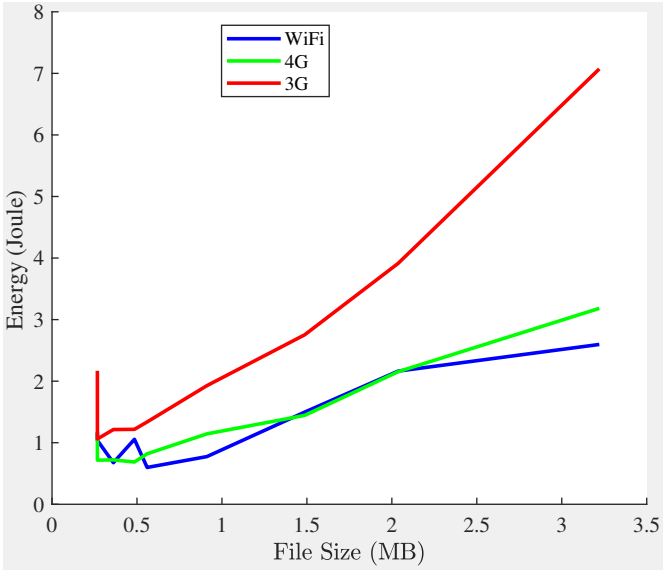


Figure 3.8: Average energy cost of uploading files of different sizes.

3.4.1 Energy Consumption

Measuring energy consumed is more complicated than measuring time, because by default the Android Operating System (OS) reports battery level in percentage only. So the resolution of reported energy is too high. Also as many processes are running simultaneously in the background, it is hard to calculate the exact energy consumed for the communication purpose. On top of that, the energy consumed by the profiler itself has to be accounted for. So a third party application called the Trepn profiler [105] is used to profile the energy consumption. More detail on Trepn Profiler and how it is used in described in Section 3.3.1. Figure 3.8 shows the average energy consumption per transmission. WiFi and 4G are the most energy efficient modality to transmit files and only consumed about 1 Joules to upload a file of 1 MB. In the experiments, 3G consumed the most energy for transmission. To transmit 1 MB, it used about twice as much energy than 4G and the WiFi. The energy gap is more pronounced as the file size increases.

Figure 3.9 shows energy consumption of every transmission for all three modalities based on file size and the radio signal strength during the transmission and their linear fits.

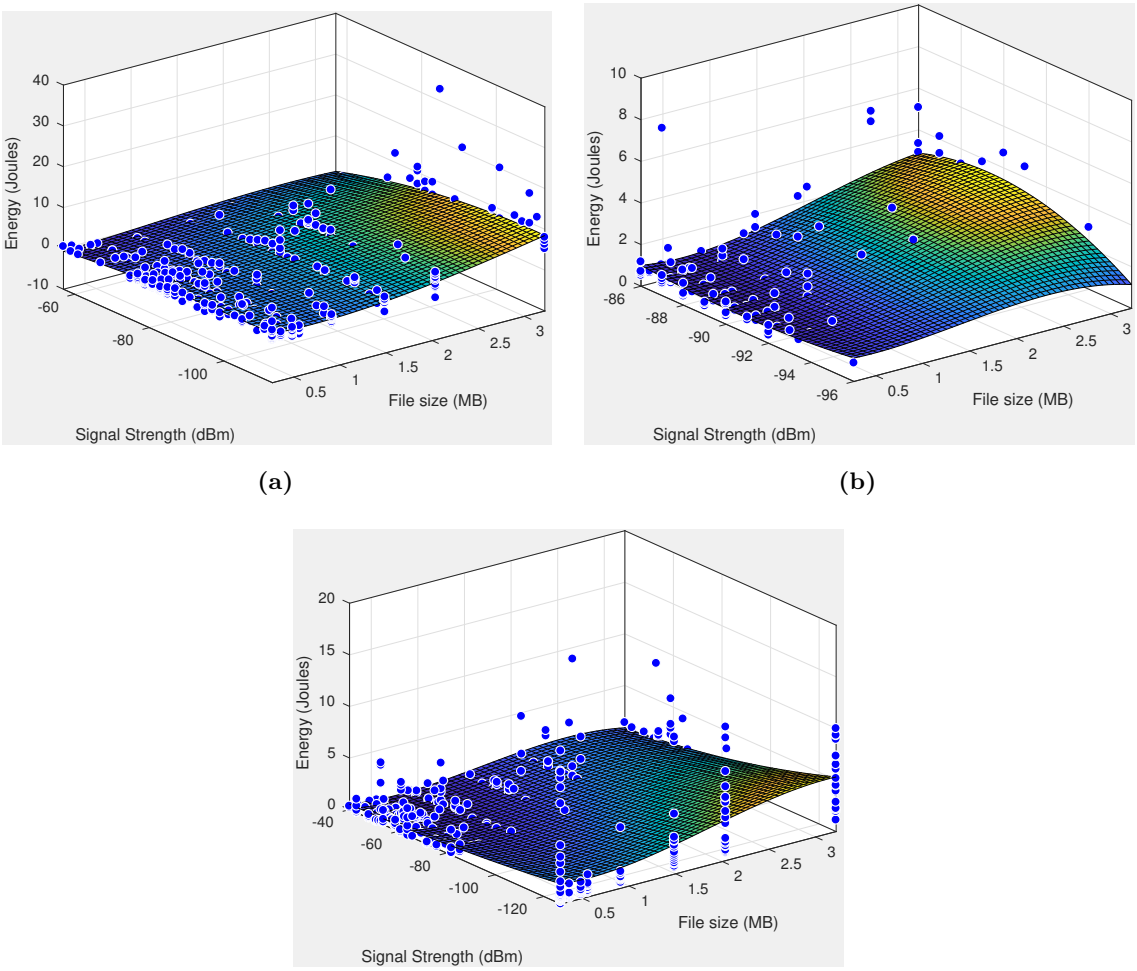


Figure 3.9: Energy required to upload files of different sizes under different communication modalities and signal strengths. (a) 3G (b) 4G (c) WiFi

The linear fits shows the energy consumption is clearly dependant on the file size and also to some extent the radio signal strength. When the signal strength is low, the Bit Error Rate (BER) would be higher, so the Packet Delivery Rate (PDR) would be lower. This means each packet has to be transmitted more than once in average which results in the higher energy consumption. The WiFi energy cost in Figure 3.9c is the most clear example of this relationship. It is slightly evident in Figure 3.9a for 3 MB file size. However, for 4G results in Figure 3.9b it seemed negligible. This is due to the fact that during the experiments, the RSSI level did not change significantly to have any pronounced effect.

3.4.2 Initial Radio States

There may be several other factors affecting the transmission times and energy consumption. One of the important factor is the radio state. The User Equipment (UE) of radio communication can be modelled using a Finite State Machine (FSM), and depending on which state the radio is in, the required time and energy used for transmission can be different. For example, if the cellular radio is in idle state, it can take up to 2 seconds to be in the fully connected state whereby transmission can take place. This will be be described in the next chapter when the smartphone device is modelled. In this experiment, random delays is introduced in between transmissions so that the communication time and energy can be calculated for starting at different states. Table 3.4 shows various radio states the UE may be in before the transmission started. During the experiments, however, it cannot be guaranteed that the device would be in certain state by introducing a delay as the device could be communicating in background for various other tasks.

Figure 3.10 shows the time taken for each transmission based on the initial states and their linear fits. For 3G and 4G in Figures 3.10a and 3.10b, the radio is already in one of the connected state (i.e. radio states $2-6$). So the time taken is indifferent although, state 6 which is "the connected and sending and receiving state" is slightly faster. In

Table 3.4: Various radio states for mobile data and WiFi in an Android device

State	Cellular(3G and 4G)	WiFi
0	Disconnected	Currently being disabled
1	Connecting	Disabled
2	Connected, dormant	Currently being enabled
3	Connected, no traffic	Enabled
4	Connected, sending traffic	–
5	Connected, receiving traffic	–
6	Connected, sending and receiving traffic	–

case of the WiFi –see Figure 3.10c, there is no significant difference noted. Obviously, the case would be different if there isn't any known WiFi hosts in the range.

The energy consumption is slightly different as seen in Figure 3.11. There is an opposite trend seen in the cellular and WiFi. For WiFi, when the radio state is *connected*, the energy consumption is lower when compared to the radio state is *disabled* at the start. However, for the cellular, the energy consumption is very slightly higher when the radio state is already receiving and transmitting as shown in the Figures 3.11a and 3.11b.

3.4.3 Summary

In this experiment, the focus is to learn about the time and the energy cost for transmitting data from the mobile device. The tests are conducted with different communication modalities such as 3G, 4G and WiFi under various channel conditions and device settings. The results show that 4G can offer the fastest transmission times whereas 3G is the slowest. However, WiFi can offer significant speed while also being the most energy efficient solution. In addition, the experiments also help to understand the effects of various channel conditions and radio states.

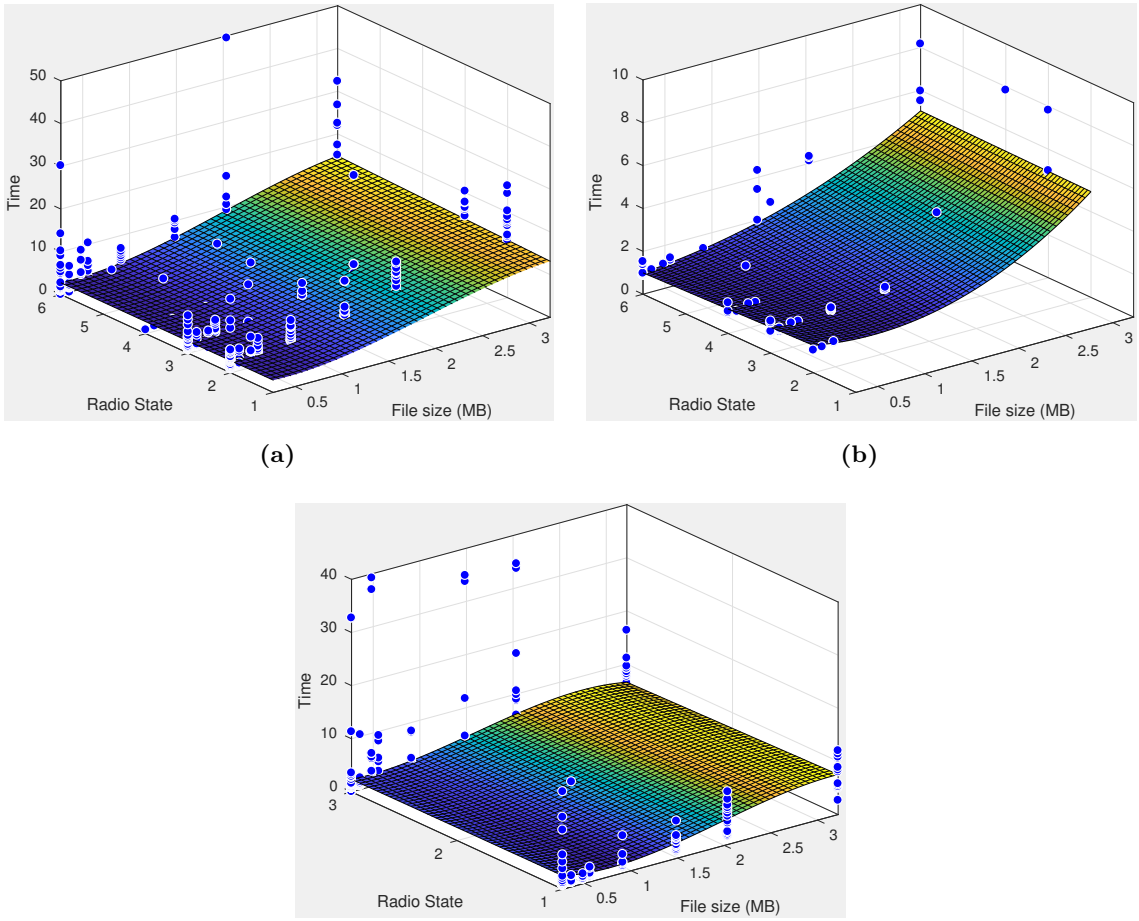


Figure 3.10: Time required to upload files of different sizes under different communication modalities and signal strengths. (a) 3G (b) 4G (c) WiFi

3.5 Discussion and Conclusion

This chapter conducted three experiments that forms a strong argument on the need and suitability of computation offloading. The first experiment in section 3.2 compared accuracy and relative time complexity of the three existing algorithms namely KISSME, SDALF, and Unsupervised Saliency. The experiments showed that the KISSME algorithm would be most suitable for distributed implementation based on its simplistic descriptor and fast training and execution times. However, many new approaches based on deep learning based approach has been proposed in the literature achieving superior

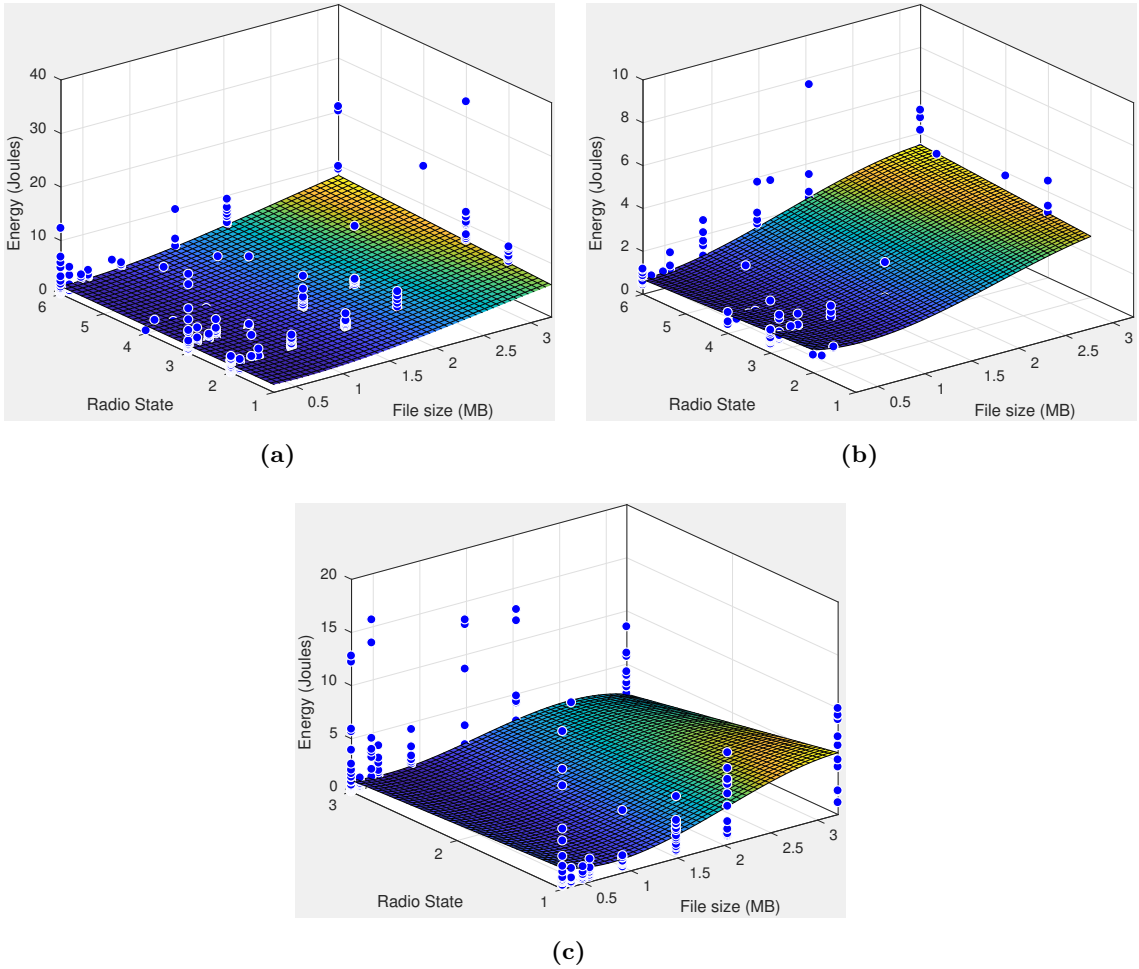


Figure 3.11: Energy required to upload files of different sizes under different communication modalities and signal strengths. (a) 3G (b) 4G (c) WiFi

results (– see Table 2.2). The second experiment showed an example of running such a deep learning algorithm on an embedded device. It demonstrates that even for the inference purpose, the DNN based approach can have significant resource usage when tasked with execution of algorithms on real-time (–see Figure 3.3). This result further reinforces the need for offloading.

The third experiment quantifies the time and energy cost while transmitting data using various modality, under different environmental conditions. Results show WiFi is an excellent communication modality that offers good communication bandwidth at a significantly low energy cost.

The three experiments show that PRID algorithms can be a good candidate for *Computation Offloading* application, especially, if the device is too busy to do it itself and bandwidth is high enough. The computation time on the device can be comparable to communicate and wait for result. In addition, the results of these experiments form a good base for the next chapter especially for designing the objective in multi-objective optimisation and detailed component based simulator design. These will be used Chapter 4 to present a framework, that can offload algorithms from embedded devices such as smartphones to other neighbouring devices as well as the cloud.

Chapter 4

Offloading Based on Multi-Objective Optimisation

4.1 Introduction

The last chapter focused on the study of existing Person Re-identification (PRID) algorithms and different challenges in its implementation on embedded devices. These approaches may be computationally expensive and take considerably long time to process on the implementation device. However, many applications require result in the real-time. One of the solutions introduced in the last chapter to achieve real-time results was *Computation Offloading*. In this chapter, *Computation Offloading* will be looked at in more detail for solving some of those issues.

4.1.1 Problem Formulation

Consider a two sensor scenario shown in Figure 4.1. Let X and Y be two similar smart cameras deployed in the field to identify people arriving in their Field Of View (FOV). Let z_{max} be the number of targets each sensor can process simultaneously in the allocated

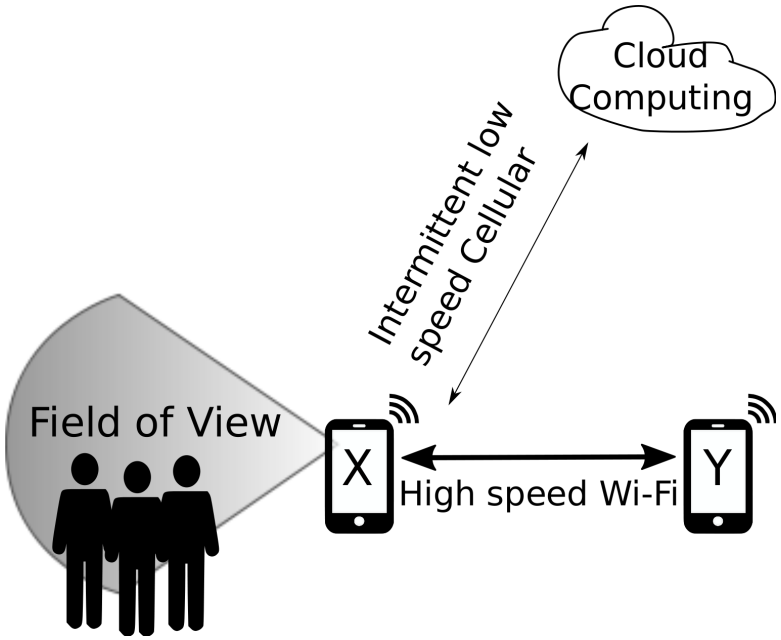


Figure 4.1: Pedestrian identification scenario: device X inundated with targets while device Y is idle

time. If each sensor is working on its own, when there are $z > z_{max}$ targets to process in sensor X, it fails to process $z - z_{max}$ targets in the allocated time-frame. Not identifying targets or dropping targets may be costly in security applications. For example, the dropped target may be a terrorist or an enemy soldier and could be life-threatening to the soldiers on the battlefield. If a connection to the cloud is available and the bandwidth is sufficiently high, the cloud can help X to process the targets in a timely fashion. Frameworks such as MAUI [106], Cuckoo [107, 108] exist that allows computation to be offloaded to the cloud. So, X can offload $(z - z_{max})$ targets to the cloud, which may save both time and energy. More on when offloading may or may not save resources is discussed in Section 4.2 If the cloud is unavailable, current systems would simply drop the targets. In this example, device Y may not have any target in its FOV at this moment and willing to help by processing some of X's targets. This chapter shows that neighbour devices like Y can be a good alternative to the cloud and may be used for *Computation Offloading*. The following assumptions are made for the PRID problem:

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 FOV, its resources (Central Processing Unit (CPU), Graphical Processing Unit (GPU)) are not fully utilised. 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.

In the next section, different platforms available for computation are defined.

4.1.2 Computing Platform Types

Onboard Computation

The onboard computation refers to the computation available on the sensor itself. When compared to a desktop computer, it may be limited however, if there is no network connectivity to the cloud, it is the only available option. Some onboard processing may reduce the amount of data that needs to be communicated while offloading. For instance, imagine X in Figure 4.1 is capturing images at 1 Frames Per Second (FPS) and sending image data frame by frame to the cloud for finding targets in the images. If there is a target, the cloud identifies the target and sends the identity of the target back to the sensor. If on average, one target appears in the image per minute, rest of the image data (59 images per minute) that is sent to the cloud is wasted. A background subtraction algorithm running on the sensor can decide if there is any foreground activity in the scene and the sensor can only send images with foreground activity. This saves communication cost for the sensor, and the cloud has fewer jobs to perform.

Cloud

In this context, the cloud may be defined as a computing platform with very high computational capability and mains powered. Researchers have successfully used the cloud to offload from the mobile devices. For example, Cuervo et al. [106] proposed energy aware algorithm to offload to the cloud for video game and chess application using WiFi and cellular connection that resulted in 27 and 45 percent energy savings. Similarly, Fernando [109] used Bluetooth to offload to a central entity. More examples of Mobile Cloud Computing (MCC) include [110–112] and further details on MCC can be found in [113–116] Also, similar to the cloud but significantly less powerful entities have been introduced lately called Fog or cloudlets [117–119]. Magurawalage et al. have considered offloading to an intermediate cloud-like entity called cloudlets [120]. Cloudlets lie between the mobile device and the cloud physically. As such the Round Trip Time (RTT) can be significantly lower compared to the cloud. However, they may be unavailable just like the cloud.

Neighbouring Nodes

Neighbouring devices lack the computational power and energy of the cloud but may be readily available with high Network Bandwidth (BW) connections. Neighbouring nodes are an important alternative to the cloud for two reasons: first is that communicating with neighbours can help in the co-ordination and control of the node network. Neighbours can provide an alert signal of an incoming target, or give complementary and valuable information (a priori) about the targets. More details about co-ordination and control can be found in [121]. The second reason is that when the cloud is unavailable, they can help in sharing the computational load. So far in the literature, neighbouring devices have not been considered for *Computation Offloading* purpose. However, as the neighbouring devices may be battery powered as well the energy impact on the neighbours cannot be ignored. This is because when in the field, charging may not be readily available.

Table 4.1: 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**

4.1.3 Summary

The main pros and cons of offloading to cloud vs offloading to neighbouring nodes are summarised in Table 4.1. The highlighted items show where each one may be superior to the other in a typical case. In case of higher bandwidth between neighbouring nodes is based on the availability of Wi-Fi among neighbouring devices whereas only low-speed cellular is available to the cloud. Only in some cases, neighbouring nodes have benefits over the cloud. However, as it was stated earlier in the thesis, the cloud may be unavailable due to several reasons such as natural disasters, terrorist attack etc.

This chapter is structured in the following way. Section 4.2 formally defines *Computation Offloading* and the motivation for sensors to use it. In Section 4.3, a PRID system is presented that poses *Computation Offloading* as a multi-objective optimisation problem. Section 4.4 proposes novel algorithms to solve them. The effectiveness of the algorithms were tested by building and testing on a simulator. In Section 4.5 the simulator is explained in detail. Then, in Section 4.6 the experimental settings and the simulations are detailed. Finally, in Section 4.7 the results are discussed and the conclusions drawn.

mobile nodes which can significantly increase the performance without substantially depleting battery resource compared to the non-offloading case.

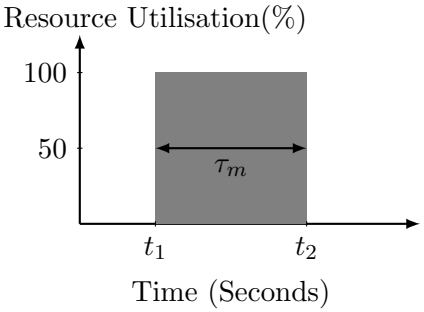


Figure 4.2: Simplistic view of resource usage during an algorithm execution. The algorithm takes τ_m seconds to execute which depends on the number of instructions required to execute for an algorithm (C) and the number of instructions that a sensor can execute every second (I_m) – see Equations (4.1) and (4.3). The gray rectangular area signifies total CPU utilisation for one algorithm execution – see Equation (4.2).

4.2 Computation Offloading

Computation Offloading can be defined as a process of executing a computationally intensive task on an alternative device rather than on its own computing platform, to save resources such as time and energy. The device that sends the task can be termed as *offloader* whereas the device that executes the task can be termed as *onloader*. Usually, *onloader*s are the devices with very high or unlimited computational capabilities and commonly known as the cloud. When the *offloader* is a mobile device and *onloader* is the cloud, the process is known as Mobile Cloud Computing (MCC). Kumar et al. [36] discussed that offloading may not always be beneficial and calculates an analytical expression for the minimum bandwidth required for the whole process to break-even in terms of energy. Even though it is fundamental and ignores real-world factors such as parallel-processing, the effect of other CPU activities and CPU speed., it is an important expression for *Computation Offloading* so, it is described below with an example.

4.2.1 Non Offloading

In order to compare the cost of offloading and its benefits, a non-offloading baseline case is considered first. Let a mobile device capable of executing I_m instructions per second run an algorithm $\mathcal{A}$, that requires C instructions to execute. Figure 4.2 shows the typical resource usage during the execution of $\mathcal{A}$. The time taken to run the algorithm τ_m can be calculated as:

$$\tau_m = \frac{C}{I_m} \tag{4.1}$$

The energy used to execute the job would be proportional to the grey area in Figure 4.2. If P_m is the power consumed by the mobile device while executing the algorithm on-board, the total energy consumption for the algorithm execution is

$$E_m = P_m \times \tau_m \tag{4.2}$$

The number of instructions required for an algorithm can change. For instance, in the Mixture of Gaussians (MOG) algorithm for background subtraction, each pixel value is compared to a number of existing Gaussian distributions to assess which distribution best represents the pixel. The number of instructions (C) depends on how quickly a matching Gaussian distribution is detected for the particular pixel [122]. In the worst case, the pixel does not match any of the existing distribution, and a new distribution has to be created. However, to make calculations easier for the simulation, the number of instructions is assumed to be constant. Similarly, the processing capability (I_m) of the processor devoted to an algorithm can be dynamic. It can depend on the current frequency of the CPU and other jobs that are running on the processor.

$$I_m \propto \frac{f}{u} \tag{4.3}$$

where f is the current clock frequency, and u is the utilisation of the CPU. On the same CPU, if the clock frequency is higher, the time taken to execute will be lower and vice-versa. Similarly, if the average CPU utilisation is higher which means there may be many jobs running on the system, then the time taken will be higher. So, Equation (4.1)

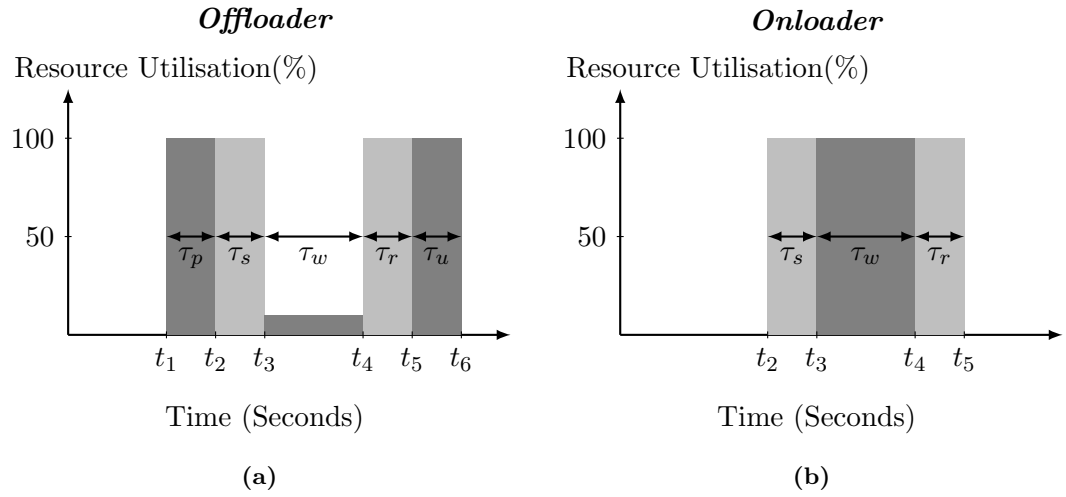


Figure 4.3: Resource usage for *offloader*(a) and *onloader*(b) during *Computation Offloading*. The gray rectangular area represents CPU utilisation whereas the light gray area represents radio usage for data communication. The times are not correctly scaled in the picture.

becomes

$$\tau_m = \frac{C}{\mathbb{E}[I_m]} \tag{4.4}$$

where $\mathbb{E}[I_m]$ is the average instructions per second available for the algorithm based on average clock frequency and utilisation of the CPU.

4.2.2 Offloading

Now, consider the offloading scenario when the task is offloaded to another device. For that, the input data to the algorithm has to be packeted and sent to the *onloader*. Then the *onloader* then processes the data and sends the result back to the *offloader*. Figure 4.3a shows the CPU and the radio usage for the *offloader*. The *offloader* can save time by offloading if the total time, referred to as makespan [123] is less than the on-board computation time:

$$\tau_{total} = \tau_p + \tau_s + \tau_w + \tau_r + \tau_u < \tau_m \tag{4.5}$$

where $\tau_p, \tau_s, \tau_w, \tau_r, \tau_u$ represent the times taken by the *offloader* to packet, send, wait, receive, and un-packet the data, respectively. If D_{in} and D_{out} are the data size to be transferred and received respectively, the communication time is given by:

$$\tau_s + \tau_r = \frac{D_{in} + D_{out}}{BW} \tag{4.6}$$

where BW is the available bandwidth. Also, if C_p be the number of instructions required for packeting one packet of data. The packeting operation can be estimated as

$$\tau_p = \frac{C_p \times \left\lceil \frac{D_{in}}{D_p} \right\rceil}{I_m} \tag{4.7}$$

where, D_p is the packet size and $\lceil x \rceil$ represents the least integer that is greater than or equal to x. The waiting time (τ_w) can be estimated similar to Equation (4.1) as:

$$\tau_w = \frac{\tau_m}{F} \tag{4.8}$$

where F is the factor by which the *onloader* is faster than the *offloader*. The speed up factor may depend on various things. For example, if the $\mathcal{A}$ has many parallelisable sub-routines it may benefit from executing on a multi-core CPU, GPU or an Field Programmable Gate Array (FPGA), rather than a single core CPU. The Amdahl's law [124] provides a simple yet effective theoretical upper limit for speed up by parallel execution and stated as:

$$max(F) = \frac{1}{r_s + \frac{r_p}{n}} \tag{4.9a}$$

$$r_s + r_p = 1 \tag{4.9b}$$

where r_s and r_p are percentage of serial and parallel code in an algorithm and n is the number of parallel executions. Similarly, if the clock frequency of the *onloader* is faster than that of the *offloader* then F can be higher than one. To keep things simple, it is only assumed that the *onloader* is F times faster.

If P_c and P_w be the average power consumption for communication and waiting times,

the total energy consumed by the *offloader* for the offloading process can be estimated as

$$\begin{aligned} E_{off} &= P_c \times (\tau_s + \tau_r) + P_w \times \tau_w \\ &= P_c \times \frac{D}{BW} + P_w \times \frac{\tau_m}{F} \end{aligned} \tag{4.10}$$

For offloading energy cost to break even with the computation cost, Equation (4.2) and Equation (4.10) should be equal, which gives rise to following equation.

$$\begin{aligned} P_m \times \tau_m - P_c \times \frac{D}{BW} - P_w \times \frac{\tau_m}{F} \geq 0 \\ \frac{C}{I_m}\left(P_m - \frac{P_w}{F}\right) - P_c \times \frac{D}{BW} \geq 0 \end{aligned} \tag{4.11}$$

For the *Computation Offloading* to save time and energy, Equation (4.5) and Equation (4.11) has to be satisfied respectively. They depend on multiple factors such as speed up factor (F), BW and the data to be communicated (D). An intuitive way of deciding whether to offload to the cloud (MCC) can be portrayed in Figure 4.4. Say the cloud is significantly powerful than the sensor device ($F = 100$) and is always available. Based on this figure, it is advisable to always offload high computation jobs requiring low data transfer (green area in the figure). For example, PRID requires image of a person (typically $<$ 1 MegaByte) as input. Based on the experiment conducted in the last chapter (see Figure 3.6), it could typically take one second to upload it using 4G and less than two seconds using WiFi. If it takes tens of second to run the algorithm on the mobile device, it is worth offloading to the cloud. Whereas, if an algorithm required several MegaBytes of data and it would only take couple of seconds to execute on mobile device, it is not worth offloading. Sometimes however, it may not be clear what is the best platform to execute the algorithm because the data to be transferred is not exactly low and the computation required is not very high. In those cases, the decision can be taken based on bandwidth. If the bandwidth to the cloud is high such that the communication can take place relatively quickly then it is worth offloading otherwise it's not.

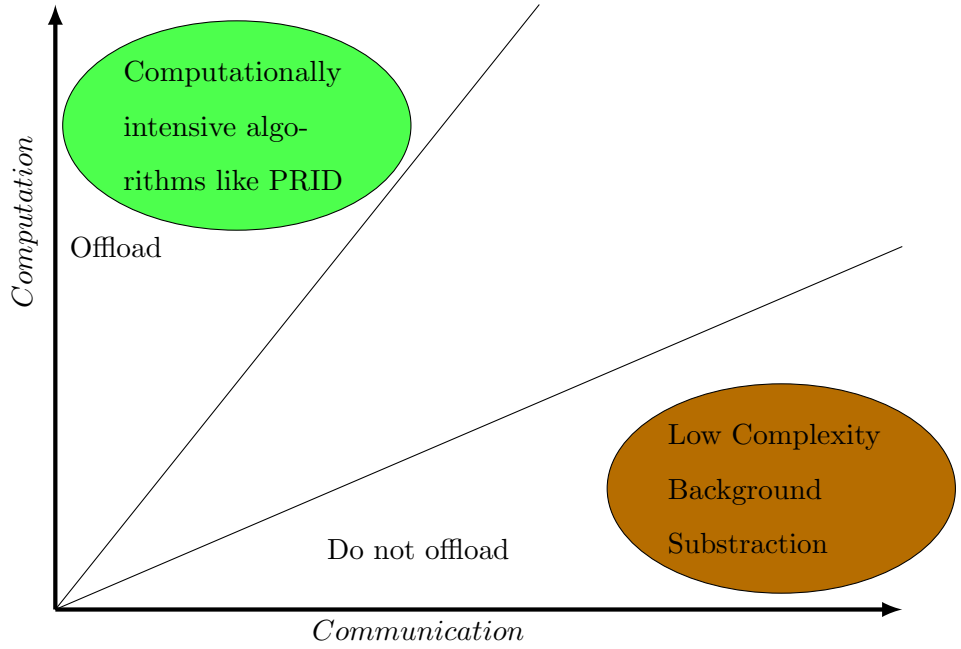


Figure 4.4: Low complexity jobs with high communication overhead should not be offloaded, high complexity jobs with low communication overhead should be offloaded and anything else depends on the bandwidth. [36]. Based on the assumption that the cloud is available at all times.

4.2.3 Impact on *Onloader*

In Section 4.2, the time and energy cost for the *offloader* was discussed. However, there is also an impact on the *onloader* as shown in Figure 4.3b. When the *onloader* is the cloud with comparably infinite computing resources and mains powered, it may be ignored. But if the neighbour is considered as an *onloader*, the impact cannot be ignored. Due to finite resources, the impact on the *onloader* is that its resources such as CPU may not be available for its own usage while it is helping the *offloader* and cost energy as well. The energy cost is similar to the *offloader* cost and can be quantised as following:

$$E_{on} = P_c \times \frac{D}{BW} + P_o \times \frac{\tau_m}{F} \tag{4.12}$$

where P_o is the instantaneous power of the *onloader* while executing the algorithm. If the *onloader* is considered to be identical to *offloader* running at same frequency and same utilisation, Equation (4.12) can be written as:

$$E_{on} = P_c \times \frac{D}{BW} + E_m \tag{4.13}$$

Comparing with Equation (4.2), the first term in Equation (4.13) is the overhead due to offloading.

4.2.4 Motivation for a Neighbour to be *Onloader*

In the previous section, general conditions were derived for offloading to be fruitful in terms of time and energy. 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. Also, the cloud is mains powered, so not limited by energy consumption. 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?"*

However, there are many environments where the cloud may not be available, or it may only be available intermittently. For example, underground train stations may not have cellular or WiFi coverage. Similarly, in the battlefield scenarios, the cloud may only be available at random times and lower bandwidths. Even if the cloud is normally available, natural disasters such as earthquake, flood and tsunami can disrupt the coverage of a large area or city making cloud inaccessible. Until now, there was not much alternative than "to try your best and hope for the best". This work purposes that even if there is limited or no access to the cloud, neighbouring devices may be able to help each other.

In Section 2.3.1, an application for anomaly detection was discussed and Figure 2.11 showed camera footage from multiple cameras. Out of four cameras, only two had foreground activity during the fighting incident. The case motivates the belief that not all the cameras in the network may be busy at the same time and may be able to help the other cameras. To formalise the problem, let's start by defining the platforms available for computation in different scenarios and their advantages and disadvantages.

The sensors can be thought of like agents in a multi-agent scenarios, that have their own beliefs, desires, and intentions [125]. It is natural to ask why would the neighbours act as an *onloader* to help others. Even though *Computation Offloading* can benefit the *offloader*, why would the *onloader* go the extra mile to help the *offloader*. However, there are several reasons why neighbours may want to help others, as discussed below,

1. The network of sensors may be cooperative sensors with a common goal, and all of them are working towards achieving the common objective. For example, in the case studies discussed in the last chapter, the common objective is to identify as many people (or targets) as possible within the whole system irrespective of how many each sensor can identify.

2. Even if the sensors are selfish and non-cooperative agents, they may still be rational, and it may still be in their interest to help others. For example, there may be an agreement between sensors before helping, that the favour would be returned in the future if needed.

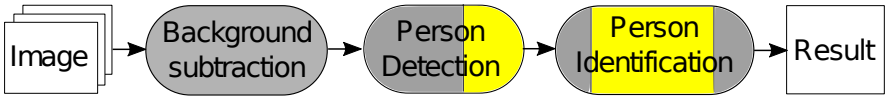


Figure 4.5: Typical pedestrian identification flowchart showing non-offloadable and offloadable sections. Gray section represents the portion of algorithm that are characterised as non-offloadable whereas the yellow section is offloadable.

3. One by-product of *Computation Offloading* is that the information can get spread out. However, it may be desirable to disseminate this information, as it may be valuable to the *onloader* as well. Consider the PRID case again, by helping neighbours in identifying the targets; the *onloader* can compile a list of targets likely to appear in their own FOV. So when any person is detected, it can search through this shorter list of targets before searching through the entire list of known targets. This may feel like added security risk especially if the application is security-sensitive operation. However, the process of *Computation Offloading* would require extra software installation, and even if other devices have access to this software, measures can be taken to only offload to the known friendly neighbours to mitigate the risks.

4.3 System Design

Given the knowledge of the cost of onboard processing and the cost of *Computation Offloading* for both the *offloader* and the *onloader*, this section aims to design a system to reap the benefits of *Computation Offloading*. As an exemplar, the pedestrian re-identification system outlined in Figure 4.1 is considered, but instead of two sensors, it is extended for S sensors. A typical workflow of a PRID system is shown in Figure 4.5 repeated from Chapter 3. It starts with image acquisition from the image sensor. A background subtraction and a person detection algorithm is applied on the image to detect the number of people in the view. When a pedestrian is detected, one of the person re-identification algorithm discussed in Section 2.4 is applied to each detection. Our goal is to identify as many detections as possible.

This chapter makes some important assumptions. The first assumption is that the targets may arrive in the FOV of the sensors at random times. In a real situation, there may be patterns of target flow which may be exploited. However, this work doesn't rely on that as the primary objective of the chapter is to show that offloading can be beneficial in the absence of cloud without any prejudice. The second assumption is that, in the set of S sensors, all the sensors have communication capabilities with every other sensor and that each sensor knows the resource information of other neighbours. This can easily be changed to communication links only between the sensors in the vicinity. This is considered in the next chapter.

As discussed in Section 4.2, to offload an algorithm, the data to be communicated and the computation required plays a vital role. For a PRID system with the processing chain shown in Figure 4.5, an assessment needs to be carried out to determine which part can benefit from offloading.

4.3.1 Application Partitioning

Generally, MCC implementations use static and dynamic application partitioning of algorithms based on profiling [126]. For the current work, the jobs are classified as *offloadable* or *non-offloadable* by design. Algorithmic details of the PRID algorithms were discussed in the last chapter, where it is seen the algorithmic complexity of the person re-identification algorithm outweighs that of other algorithms in the chain (see Table 4.2). So, only the PRID algorithm is considered to be *offloadable* as for the other stages in the processing chain, the communication costs and the time delay outweighs the benefits of offloading. Due to that, the overall complexity of the system can be estimated as $\mathcal{O}(N)$ where N is the number of people detected. Hence in the thesis, Offloadable algorithms and targets are used interchangeably.

Offloading an algorithm entails sending input data, waiting for the onloader to execute, and receiving output data. Before transmitting however, the data has to be formatted

Table 4.2: Execution times for CPU running @ 998.4 MHz

Algorithm	**Time**	**Input Data**	**Output Data**
Background Subtraction	0.1	$800 \times 600 \times 3$	800×600
Person Detector	0.2	$800 \times 600 \times 3$	$T \times 4^a$
Person Re-identification	5.1	$52 \times 120 \times 3^b$	

[a]T is number of targets & 4 is for position of bounding box

[b]Variable size depends on resolution of target

in packets and some overhead will be added to the processor. These operations can be a few hundred instructions per packet which needs to be added to the CPU workload [127].

4.3.2 Multi-Objective Optimisation

After partitioning the application into *offloadable* or *non-offloadable* parts, the decision problem of offloading can be posed as a multi-objective optimisation problem. A general multi objective problem can be defined as

$$min[f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_n(\mathbf{x}),] \tag{4.14a}$$

$$\mathbf{x} \in Co \tag{4.14b}$$

$$Co = \{\mathbf{x} \in \mathbb{R}^m : h(\mathbf{x}) = 0,\ g(\mathbf{x}) \geq 0\} \tag{4.14c}$$

where $n > 1$ [128]. The objective of such a multi-objective optimisation problem is to find $\mathbf{x}$ that simultaneously minimises the functions $f_i(\mathbf{x}), \forall i = 1, ..., n$. This may not always be possible when the objectives are conflicting. So in multi-objective optimisation problems, a Pareto optimal solution is generally considered. A solution $\mathbf{x}^* \in Co$ is called a Pareto Optimal solution for a multi-objective problem if all other vectors $\mathbf{x} \in Co$ have a higher value for at least one of the objective function $f_i(\mathbf{x})$, with $i = 1, ..., n$.

The cost of an algorithm execution can be expressed as multi-objective function that can be scalarized as a weighted sum of time cost and energy costs. Let s, t be two sensors in set S containing $|S|$ sensors. The cost of executing an algorithm originating at s on

sensor t can be calculated as follows:

$$Cost_{st} = w_t \times \tau + w_{st} \times E_s + w_{ts} \times E_t \tag{4.15a}$$

$$\text{where, } \begin{cases} \tau = \tau_m,\ E_s = E_m,\ E_t = 0, & \text{if } s = t \\ \tau = \tau_{total},\ E_s = E_{off},\ E_t = E_{on}, & \text{if } s \neq t \end{cases} \tag{4.15b}$$

and w_t, w_{off} and w_{on} are the weights for each objective; $\tau_m, \tau_{total}, E_m, E_{off}, E_{on}$ are the time and energy costs from Equations (4.2), (4.4), (4.5), (4.10) and (4.13). When the cloud is available, it is also considered as one of the nodes with no energy constraint. So $w_{ts} = 0$ if t is the cloud.

4.4 Algorithms

4.4.1 Minimal Energy Cost (MEC)

The cost function involves adding time and energy variables (i.e. different units), which requires careful selection of the weights. One way to avoid this situation is by limiting one of the objectives to a threshold (ε) and optimising rest of the objectives. This method is called ε constraint method and was first introduced by Chankong and Haimes [128]. This method can be thought of as opposite of Lagrange relaxation [89,129] whereby a constraint is moved to the objective function using a Lagrange multiplier. Regarding the real-time nature of the problem, a constraint is set to the time cost and so that the objective function in Equation (4.15a) only contains the energy variables. Equation (4.15a) then becomes

$$Cost_{st} = w_{st} \times E_{st} + w_{ts} \times E_{ts} \tag{4.16a}$$

$$\text{subject to, } \tau \leq \varepsilon \tag{4.16b}$$

For this problem, the ε is set to 25 seconds and the nodes that do not satisfy this constraint are left out. This is highlighted in Figure 4.6) by light dots. A set $\mathcal{F}$ is created that

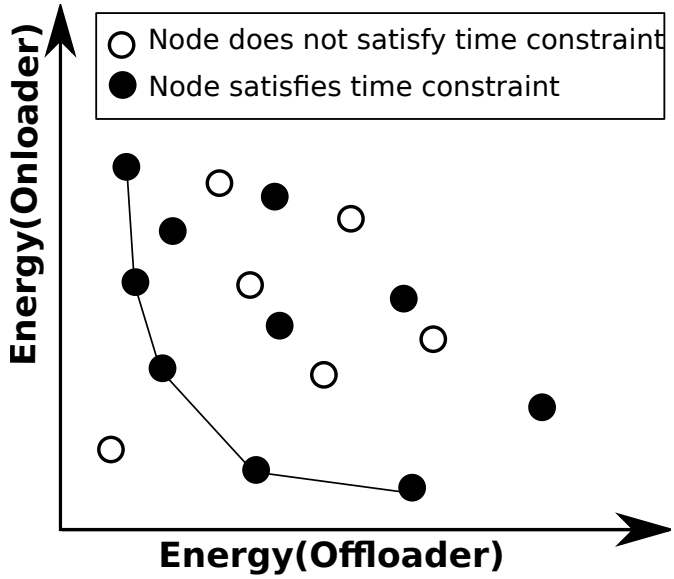


Figure 4.6: Multi-Objective optimisation problem reduced to two objectives. Dark dots represent nodes satisfying the constraint in Equation (4.16b). The solid line represents the Pareto Optimal curve

contains only the nodes that satisfy the constraint (dark nodes in Figure 4.6). Without losing generality, the set is then ordered such that $Cost_{s1} \leq Cost_{s2} \leq, ... \leq Cost_{s|F|}$. The first node on the list is picked for execution of the algorithm. The algorithm is detailed in Algorithm 2. Running the optimisation algorithm every time there is a target may not be feasible. In that case, the ordering can be created once and instead of creating a new list everytime, incremental changes can be used to shuffle and modify the sensor list.

4.4.2 Minimal Battery Impact (MBI)

In a battery-powered device, using the least energy cost per job alone may not be sufficient to increase device lifetime. For example, say an algorithm costs 10 and 8 Joules on devices X and Y, respectively. But X and Y have 500 and 50 Joules left in their battery respectively. Considering energy cost alone, Y is the best choice but when the amount of energy left in the device clearly X is a better choice. So, in this method, the Equation (4.16a) in MEC is slightly changed to such that instead of finding the node

Algorithm 2: Minimal Energy Cost (MEC) and Minimal Battery Impact (MBI)

Input: Set of sensors(S), Set of offloadable algorithms(Λ), $Cost$

Parameter: Threshold time ε

Output: s_1

```
for s ∈ S do
    for a ∈ Λ_s do
        Estimate the Cost_st of execution in each sensor in the F based on
         Equation (4.16a) or Equation (4.17a).
        Create a set F ⊆ S such that τ_f ≤ ε, ∀s ∈ F
        Order F in the ascending order of execution cost such that
         Cost_s1 < Cost_s2 <, ... < Cost_s|F|
        Select the first sensor in the list for execution.
    end
end
```

with the lowest energy cost, a node with least impact is chosen.

$$Cost_{st} = w_{st} \times \frac{E_{st}}{E_{s_{rem}}} + w_{ts} \times \frac{E_{ts}}{E_{t_{rem}}} \tag{4.17a}$$

$$\text{subject to, } \tau \leq \varepsilon \tag{4.17b}$$

where $E_{s_{rem}}$ and $E_{t_{rem}}$ are the energy left in the *offloader* and the *onloader* nodes. The algorithm is detailed in algorithm 2 as well.

4.4.3 Offload Only if Busy (OOB)

The previous methods try to find a global solution, but offloading has overhead costs. So, this method tries to offload only if on-board processing is estimated to be infeasible. Based on how many algorithms are ahead in the queue and using Equation (4.4) the processing time is estimated. If this time is greater than the threshold (ε), then offload the algorithm minimising the time and energy objectives defined Equation (4.15a). The algorithm is detailed in Algorithm 3.

Algorithm 3: Offload Only if Busy (OOB)

Input: Set of sensors(S), Set of offloadable algorithms(Λ), $Cost$

Parameter: Threshold time ε

Output: s_1

```
for s ∈ S do
    for a ∈ Λ_s do
        Estimate τ_m for self execution.
        if τ_m ≤ ε then
            Execute locally on s
        else
            Remove s from set S to create F
            Estimate the cost of execution in each sensor in the F based on
             Equation (4.15a).
            Order F in the ascending order of execution cost.
            Cost_s1 < Cost_s2 <, ... < Cost_s|F|
            Select the first sensor in the list for execution.
        end
    end
end
```

4.5 Simulator

There are two ways to test the algorithms in Section 4.4. Either it can be implemented on a number of smart-phones and tested on the real test scenarios or, use a simulator. While there are many network simulators such as NS-2 [130], NS3 [131] and OPNET [132], there is a scarcity of a holistic simulator simulating both computation and communication. In a recent article [133], the authors talk about the importance of a holistic simulator for smart-camera network and present their discrete event-based simulator. Another simulator Castalia [134] is a simulator for wireless sensor network. They specialise on communication between devices. The existing simulators are good when designing network protocols and application. However, as they are discrete event simulator, it is difficult to attribute time and energy calculation by each component of devices. The power model used in the existing simulators are rudimentary and not sufficiently fine grained.

Commercial off-the-shelf (COTS) smart-phones are sophisticated devices, and their performance is subject to hundreds of parameters and constraints. Some of these parameters may be tuned whereas others can only be observed. For instance, the FPS of the image sensor may be changed as required whereas the clock frequency of the CPU and its utilisation may not be available for selection and depend on the Operating System (OS) Kernel. A robust simulator is required that can emulate the execution of algorithms and communication between devices. The major elements of the proposed simulator relate to the algorithmic tasks, the sensor architecture, communication links and the targets. Implementation of the simulator is explained using class diagram in Appendix B Each one is described in detail below.

4.5.1 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 energy consumption of the display is ignored as it can be assumed to be turned off by the application. In the literature, there are

many power models that can be used to estimate the power consumption of individual component of the smart-phone [135–140]. A comprehensive list of power models can be found in the thesis [141]. For this simulator, the utilisation based model by Jung et al. is used because of the simplistic model [140] and the parameters are based on a Google Nexus I phone which is one of their Device Under Tests (DUTs). However if desired, the simulator can be calibrated for a different DUT in a straightforward manner.

Image Sensor

The image sensor consumes significant energy in a mobile device when used continuously. According to Likamwa et al., the energy consumption per frame of the image sensor can be modelled as follows [142].

$$E_{\text{camera}} = P_{\text{idle}} \times (T_{frame} - T_{active}) + P_{active} \times T_{\text{active}} \tag{4.18}$$

where $T_{\text{active}} = \frac{\text{Number of Pixels}}{\text{Camera Clock Frequency}}$. The Camera clock is separate from the CPU clock and generally operate at around 32 MHz. Figure 4.7 shows the power consumption during sequential capture mode. The power may depend on several factors such as the FPS determines the length of T_{frame}. Also, based on Equation (4.18), the image resolution and the frequency can also change T_{active} thereby affecting the energy consumption.

Application Processor (AP)

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

$$p^{\text{cpu}} = \beta^{\text{cpu}}_{\text{freq}} \times u + \beta^{\text{cpu}}_{\text{idle}}, \tag{4.19}$$

where u is the utilisation and $\beta^{\text{cpu}}_{\text{freq}}$ and $\beta^{\text{cpu}}_{\text{idle}}$ are the CPU parameters. All the parameter values are listed in Appendix A. The relationship between CPU power, utilisation and CPU frequency is visualised in Figure 4.8b. The utilisation is estimated as the ratio of the CPU time used to the time available per frame. However, the CPU is also used

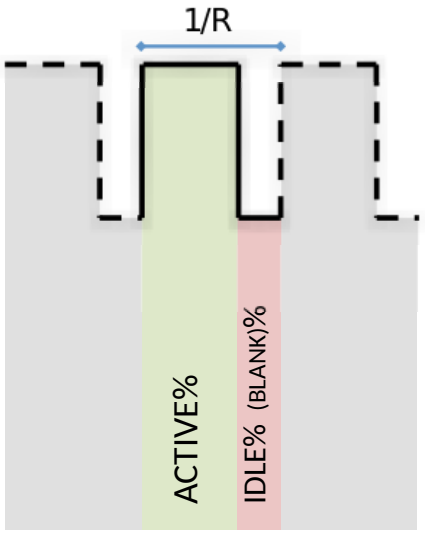


Figure 4.7: Power consumption of an image sensor in a smart phone during sequential capture mode [142]. Image from [142]

.

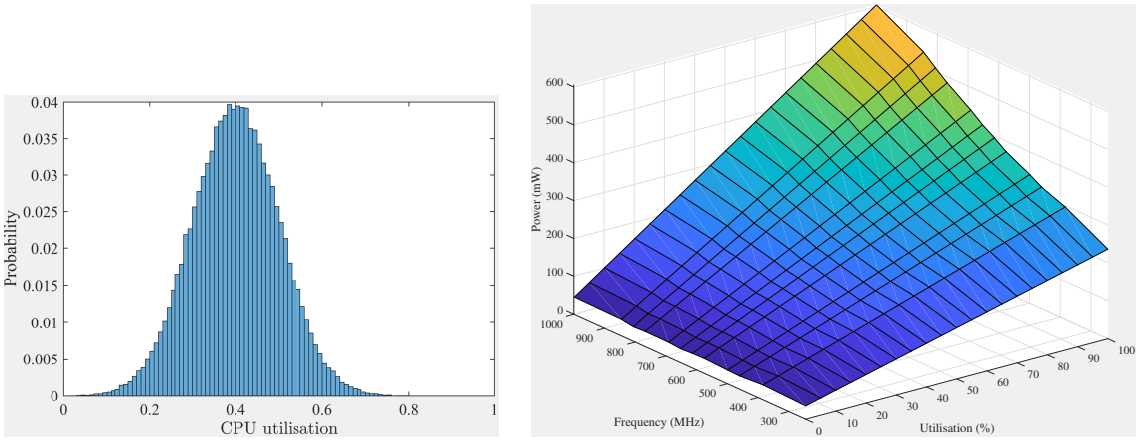
by the OS and other running applications. Dargie used a Normal and exponential distributions to simulate workload in [143]. In this work, a Gaussian process is used to simulate these other activities – See Figure 4.8a. By adjusting the mean of g a busy sensor and an idle sensor can be simulated. The total utilisation is calculated as shown below.

$$u = \frac{\sum_{i=1}^{z} T_{\text{exec}i}}{T_{\text{Frame}}} + g \tag{4.20}$$

where z is the number of algorithms to be processed, T_{exec_i} is the execution time for i^{th} algorithm (see Table 4.2 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.

Cellular (3G)

Cellular radio is modelled as a three state system: IDLE, Forward Access Channel (FACH) and Dedicated Channel (DCH). The IDLE mode is the non communicating



(a) **(b)**

Figure 4.8: (a) Probability distribution of g based on Gaussian distribution with mean of 0.4 and variance 0.1 (b) CPU power relationship with CPU frequency and utilisation.

mode and has the lowest power consumption. In this mode, the User Equipment (UE) is turned on but has not established Radio Resource Control (RRC) connection with the Radio Network Controller (RNC). In the DCH state the UE has a dedicated transport channel for data transmission in both directions, but this is 50 to 100% more expensive than FACH, where FACH is the intermediate state with reduced power consumption and low data rate. There is no dedicated channel allocated in this mode and it can only transmit user data through shared low speed channel that is typically less than 15kbps [144]. The power is only dependent on state but not on utilisation. i

$$p^{3g} = \begin{cases} \beta_{IDLE} & \text{if RRC state is IDLE} \\ \beta_{FACH} & \text{if RRC state is FACH} \\ \beta_{DCH} & \text{if RRC state is DCH} \end{cases} \tag{4.21}$$

where RRC is the current state of UE and β_{IDLE}, β_{FACH} and β_{DCH} are based on [140]. Figure 4.9 shows the state diagram with the inactivity timers which along with data buffer size controls the state promotions and demotions. When the buffer is greater than an arbitrary size, the state is promoted from IDLE to FACH and FACH to DCH.

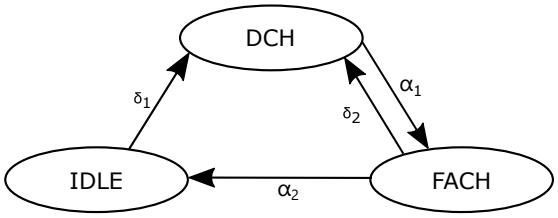


Figure 4.9: Cellular radio states, α_1 and α_2 are inactivity timers whereas δ_1 and δ_2 are delay to get to DCH

These state promotions take δ_1 and δ_2 seconds. Similarly, when the buffer size is empty ie. radio is inactive, states are demoted from DCH to FACH and FACH to IDLE. In some designs, manufacturer chose to demote from DCH to IDLE rather than visiting the intermediate state FACH [144]. These demotions, are delayed using inactivity timers α_1 and α_2 as shown in Figure 4.9 so that if any data needs to be transmitted, it doesn't need to wait for the promotion times δ_1 and δ_2.

Choosing the value of inactivity timers (α_1 and α_2) can also affect the battery life; larger value incurs less promotion delays if new data needs to be transmitted but uses more power as the UE is in higher power state for longer times whereas, smaller value means if new data needs to be transmitted, it has to wait for the promotion times incurring larger delays in transmission.

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.

$$p^{\text{wifi}} = \begin{cases} \beta_{\text{LT}} \times p + \beta_{\text{LT base}} & \text{if } p \leq \text{Threshold} \\ \beta_{\text{HT}} \times p + \beta_{\text{HT base}} & \text{if } p > \text{Threshold} \end{cases} \tag{4.22}$$

where p is the packet rate, $\beta_{\text{LT}}, \beta_{\text{HT}}, \beta_{\text{LT base}}$ and $\beta_{\text{HT base}}$ are the parameters of the DUT based on [140]. 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. Although Wi-Fi consumes significant energy in the scanning mode where it scans the environment to establish the network connection, it is ignored as a connection between the sensors is considered as the basis of this research.

4.5.2 Energy Saving Methods

A COTS mobile device implements many energy saving methods as their battery is limited. Some of them plays a vital role in the sensor lifetime and is implemented in the simulator to make the simulation realistic as much as possible.

Dynamic Frame per Second

Energy can be saved by decreasing the number of FPS of the system (see Equation (4.20)). However, very low FPS may mean some of the detections may be missed. A simple algorithm to vary the FPS of each individual sensor between 1 and 16 in the following way.

$$\text{FPS (new)} = \begin{cases} \text{FPS (old)} \times 2 & \text{if } t < \tau \\ \text{FPS (old)} \div 2 & \text{if } t > \tau \end{cases} \tag{4.23}$$

where t is the time between target activities and τ is 5 seconds.

Dynamic Voltage and Frequency Scaling (DVFS)

The CPU voltage and the clock frequency can be changed dynamically in order to save energy. Lower Voltage and frequency correspond to lower energy and lower performance and vice versa. According to Guérout et al. DVFS has been present in the Linux kernel since 2001 [145]. They are presented in Table 4.3. In this work, only the clock frequency can be changed. Also even though the frequency can range from hundreds of MHz to GHz, in practice, there are only tens of discrete values that are available.

Table 4.3: Different Frequency governors found in modern OS

Name	Description	Fixed	Frequency
Performance	Frequency is set to maximum.	✓	f_{max}
PowerSave	Frequency is set to the minimum.	✓	f_{min}
Userspace	Frequency is set to the user-specified frequency.	✓	f_{user}
Conservative	Controlled using two thresholds, frequency is changed one step at a time.		$\begin{cases} f_{current} + 1, \ if \ \mu > up_{th} \\ f_{current} - 1, \ if \ \mu < down_{th} \end{cases}$
OnDemand	Aggressive governor, goes to maximum frequency in one step but cools one step at a time		$\begin{cases} f_{max}, \ if \ \mu > th \\ f_{current} - 1, \ if \ \mu < th \end{cases}$

For the DUT for this work, the available frequency values are given in Appendix A. There are five different types of CPU frequency governors based on how they react to CPU loads. Three of them are static, Performance, PowerSave and Userspace sets the CPU clock frequencies to the highest available, lowest available and user specified value respectively. The conservative mode has two thresholds, up_{th} and $down_{th}$. Whenever the CPU utilisation reaches one of those thresholds, the frequency is changed one step at a time. Lastly the OnDemand works with only one threshold (th). Whenever that threshold is exceeded, the OnDemand sets the CPU frequency to the maximum straight away. However, lowering the frequency is similar to the conservative mode i.e. one step at a time.

Table 4.4: Frequency governor selection based on remaining battery level percentage

Battery Percentage	Frequency Governor
0 – 20	PowerSave
20 – 50	Conservative
50 – 80	OnDemand
80 – 100	Performance

Active Power Management (APM)

By default, the Performance governor is selected in all the sensors in the network. So, all the sensors are running their CPU at the highest available frequencies. When APM is enabled, the APM selects the frequency governor based on the remaining battery in the sensor device – see Table 4.4.

4.5.3 Targets

In this work, the targets are generated using Poisson Point Process (PPP) and follow the random waypoint model in [92] and described in Section 2.5.3. In this model, targets move from one point to another with random speed and can also pause for a random amount of time. This is described in Section 2.5.3. When they enter into the FOV of the sensors (shown as orange and yellow areas in Figure 4.10), they are tagged as detected targets. The detected targets are then identified. Once the target has been detected and identified, it does not have to be re-identified again in the same camera. This is done to simulate tracking the target in each camera and using the best representation for identification purposes.

4.5.4 Metric – Efficiency Score (ES)

In order to assess the performance of the proposed algorithms, two approaches are used. First, each algorithm's performance is compared against the Non Offloading (NO)

Table 4.5: Three Algorithm "A", "B" and "C" to choose from, to do same work. Which one is better?

Sensor	**Total Jobs**	**Jobs Completed**			**Energy Used**		
		A	**B**	**C**	**A**	**B**	**C**
I	8	4	5	7	20	22	30
II	8	4	7	7	30	26	40
Total	16	8	12	**14**	**50**	58	70

case. The *process score* is defined as the percentage of jobs successfully executed in the allocated times. Secondly, a new simple metric called ES is introduced as the ratio of *Successful Identification* to the energy consumed. Similar metric (mAP/Energy) has also been used by Mao et.al [146] 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 ES of an algorithm can be defined as:

$$\text{ES} = \sum_{i=1}^{\text{Nodes}} \frac{\text{No. of jobs completed}_i}{\text{Energy Used}_i} \tag{4.24}$$

ES should not be confused with the accuracy and energy consumption of the PRID algorithm. Lets take an example to see how it could be used. A system has two sensors, each presented with eight jobs – see Table 4.5. There are three algorithms available "A", "B" and "C". "A" processes the least number of jobs but consumes the least energy. "C" processes the most number of jobs but also consumes the most amount of energy. "B" is in the middle of the spectrum between "A" and "C". Given a choice between "A", "B" and "C" which is one is the best or the most rational choice? The answer to that may depend on number of things.

For instance, if there is no limitation to the energy consumption, algorithm "C" is the best choice as it can accomplish the most number of jobs. Similarly, if the sensor life is to be extended as much as possible, "A" should be selected as it uses the least amount of energy. However, if the energy is limited, the answer may not be straightforward. So,

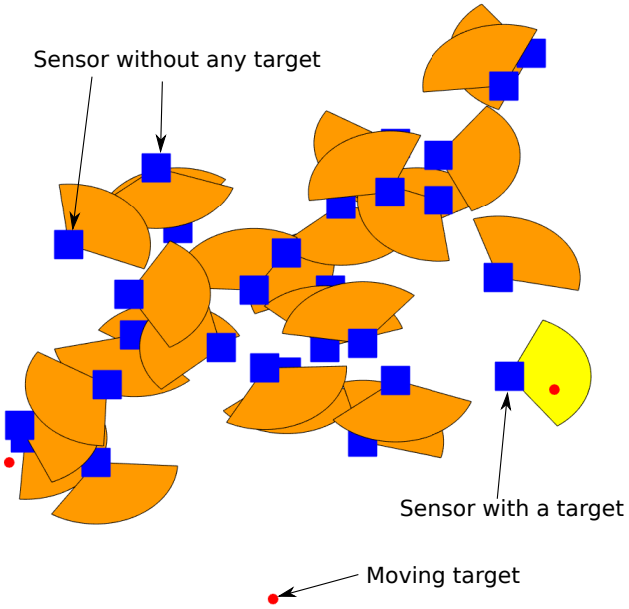


Figure 4.10: Snapshot of simulation showing targets as dots and sensors. (Yellow FOV signifies target in FOV whereas orange signifies no target in FOV).

a new metric is introduced that can help to make this decision easier. Based on this metric, algorithms "A", "B" and "C" have scores of $0.01, 0.0129$ and 0.0125 respectively. So, "B" gives the best balance of performance vs energy consumption. In the next sections, various scenarios are explored to see if the algorithms presented in Section 4.4 work and if they is any better choice depending on the scenario.

4.6 Simulation and Results

The simulator contains a number of sensors connected to each other by Wi-Fi and to the server (when available) by a cellular link. Figure 4.10 shows a snapshot of the simulation, where the blue squares and red dots represent the sensors and targets respectively. For simplicity, it is assumed that the resource information about all the nodes such as remaining energy and current CPU load., is available and all the sensors have same computational capability but the server is 10 times more powerful. Also there is no

Table 4.6: Result Summary for all the cases. Keys: S=Selfish, C=Cloud Available, APM = Active Power Management Det.=Targets Detected, O=Targets offloaded, Dro.= Targets Dropped, SI = Successful Identifications, E = Energy Used, ES= Efficiency Score per 100 Joules

Algo.	**S**	**APM**	**C**	**Det.**	**O**	**Dro.**	**SI**	**E**	**ES**
NO				18.89	0.00	4.46	14.43	557.82	2.59
MEC				18.89	18.89	1.18	17.71	563.96	3.14
MBI				18.89	18.89	1.18	17.71	563.96	3.14
OOB				18.89	8.05	1.72	17.17	561.93	3.06
NO		✓		18.90	0.00	6.81	12.10	510.08	2.37
MEC		✓		18.90	18.90	4.03	14.87	516.72	2.88
MBI		✓		18.90	18.90	4.03	14.87	516.72	2.88
OOB		✓		18.90	9.41	4.50	14.41	514.00	2.80
NO	✓	✓		18.90	0.00	6.81	12.10	510.08	2.37
MEC	✓	✓		18.90	9.57	3.53	15.37	517.88	2.97
MBI	✓	✓		18.90	17.63	11.09	7.81	489.73	1.60
OOB	✓	✓		18.90	9.41	4.50	14.41	514.00	2.80
NO			✓	18.89	0.00	4.46	14.43	557.82	2.59
MEC			✓	18.89	18.89	0.00	18.88	584.14	3.23
MBI			✓	18.89	18.89	0.00	18.88	584.14	3.23
OOB			✓	18.89	6.82	0.26	18.63	572.67	3.25
NO		✓	✓	18.90	0.00	6.81	12.10	510.08	2.37
MEC		✓	✓	18.90	18.90	0.09	18.81	535.16	3.51
MBI		✓	✓	18.90	18.90	0.09	18.81	535.16	3.51
OOB		✓	✓	18.90	7.70	1.49	17.41	528.23	3.30
NO	✓	✓	✓	18.90	0.00	6.81	12.10	510.08	2.37
MEC	✓	✓	✓	18.90	18.47	0.06	18.84	535.21	3.52
MBI	✓	✓	✓	18.90	18.89	0.27	18.62	534.13	3.49
OOB	✓	✓	✓	18.90	7.70	1.49	17.41	528.23	3.30

energy limitation for the cloud so the weight $w_{ts} : t = cloud$ for the cloud is set to zero. Wi-Fi is set to 10 Mbps (high BW) whereas cellular is slower and set to 1 Mbps.

At the start of the simulation, the battery level is uniformly distributed between $0 - 10$ Watt-hour. The mean of g in equation (4.19) is uniformly distributed from $0 - 1$ (full load) and the standard deviation is fixed to 0.1 and these parameters do not change during the simulation. Full simulation data and parameters are available in Appendix A and all the results are listed in Table 4.6

4.6.1 Unselfish and Homogeneous Sensors

There are many cases when all the sensors in the group are equal in terms of computing power and cooperative with each other. For instance, in a swarm of similar drones flying to gather visual data, the drones may not have any APM and their CPU clock frequency may be constant. Their flight time is fixed at 30 minutes and mostly dependant on the energy used by the motors for keeping it airborne rather than the energy for computing. No one has any motivation to be selfish and save their battery.

In the simulator, non selfishness is ensured by selecting $w_{st} > 0$ in Equations (4.15a), (4.16a) and (4.17b). Likewise, homogeneity is maitained by turning the APM module off. The sensors operate at the highest frequency available irrespective of any other settings. Results are compared with the non offloading case and presented in the Figure 4.11. The NO case processes slightly less than 80% of the targets. When the cloud is not available, Figure 4.11a shows all three algorithms gain about $18 - 23\%$ in performance in comparison to the NO case. However, there is also some extra energy consumption due to *Computation Offloading* but this is very limited compared to the performance gain – see Figure 4.11b. When the cloud is available, most of the targets are identified, however, it comes at slightly more energy – see Figure 4.11d. Comparing between the three proposed algorithms, MEC and MBI performs slightly better than OOB. This is also confirmed by a higher ES score of 0.0314 compared to 0.0306 of OOB and 0.0259 of NO case.

4.6.2 Unselfish and Heterogeneous Sensors

This is the type of battlefield scenario described in the Section 1.3; there are many heterogeneous devices deployed but all belonging to same individual or company. To simulate the heterogeneity in the simulator, the APM is switched on. As the battery level is uniformly distributed at the start, some sensors will be running at performance mode whereas some sensors will be running in the power save mode (Section 4.5.2). Sensors running in power save mode can execute fewer instructions per second compared to the

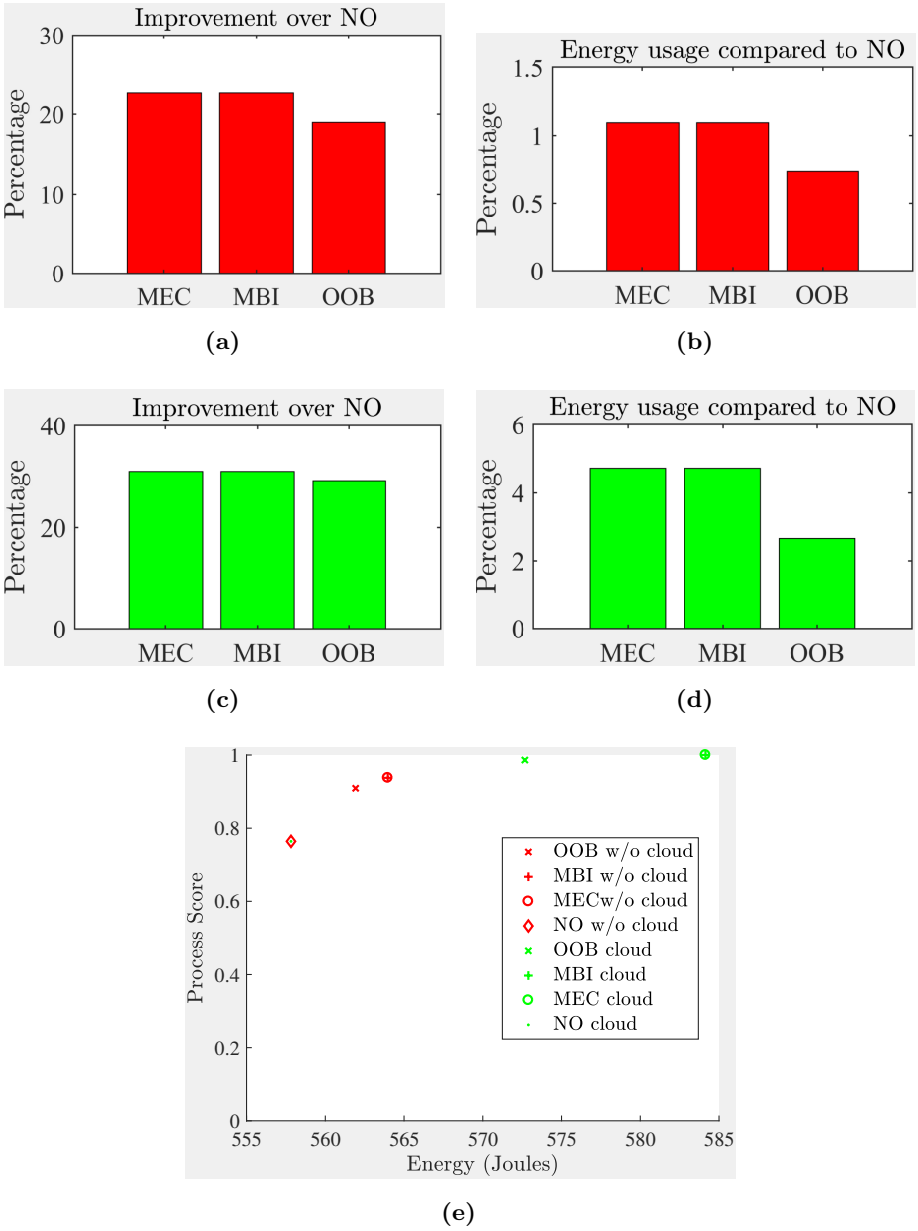


Figure 4.11: Algorithm performance improvement and energy usage for unselfish and homogeneous sensors. (a) Performance Increment and when cloud is not available. (b) Extra energy usage compared to NO when cloud is not available. (c) Performance Increment when cloud is available. (d) Extra energy usage when cloud is available. (e) Process Score vs Energy usage.

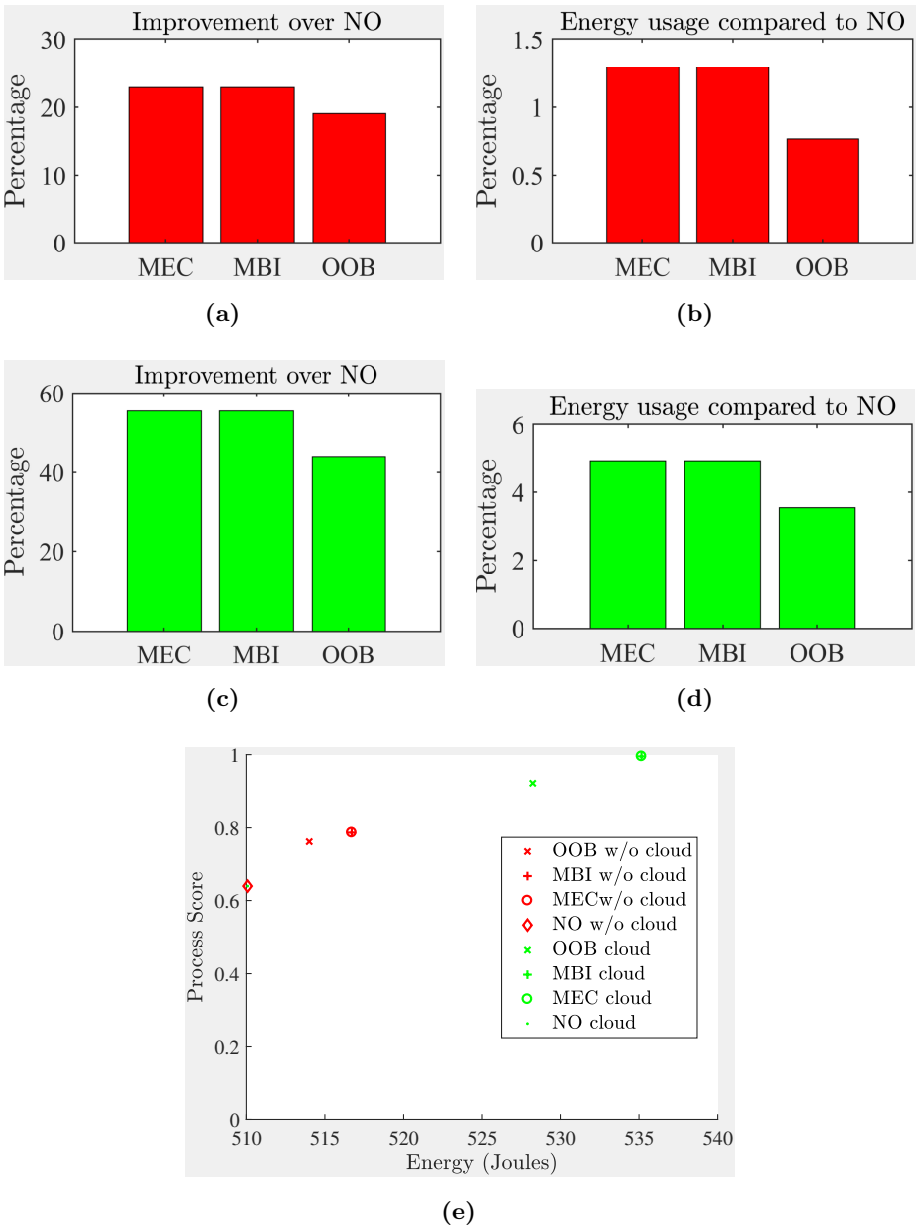


Figure 4.12: Algorithm performance for unselfish and Heterogeneous sensors (a) Performance Increment and when cloud is not available. (b) Extra energy usage compared to NO when cloud is not available. (c) Performance Increment when cloud is available. (d) Extra energy usage when cloud is available. (e) Process Score vs Energy usage.

sensors running at performance mode. This creates the heterogeneity in the sensors deployed. This is evident by the lower performance in the NO case. Only about 60% of the targets are identified by the naive NO case. The results are shown in Figure 4.12. When the cloud is not available, Figure 4.12a shows that all three proposed algorithms around 21% compared to the NO case. The OOB is slightly worse but the energy cost is slightly less in that case. When the cloud is available, Figure 4.12c shows that MEC and MBI are can gain more that 50% performance, whereas OOB only gains around 40%. This is due to the strategy of OOB that it will only offload if it cannot perform the job by itself within the allocated time. The ES of MEC and MBI is higher than that of OOB and NO at 0.0288 and 0.0351 when the cloud is not available and when the cloud is available – see Table 4.6.

4.6.3 Selfish and Heterogeneous Sensors

This is the last scenario and the most general of all scenarios. In this scenario, the sensors are selfish and the APM is enabled as well. This type of scenarios can be experienced in day to day life. For example, in a busy city centre or train station, there are thousands of people, each carrying a mobile phone. The phones can be different make, model and correspondingly can have higher or lower computational capabilities. As they belong to different individuals, they do not share any common objectives. They are only behaving to utilise the neighbours for their own benefit. The results are shown in Figure 4.13. When the cloud is available, it is the favourable option and the three proposed algorithms behave similar to the non-selfish scenarios described above. When the cloud is not available, MEC and OOB still have positive effect and gain about $20 - 25\%$ than the NO case. However, interestingly the MBI was worse than the NO case, even though it used slightly less energy. This is a result of combination of several factors, for example the bandwidth among the nodes is high so the communication cost is low, and when combined with the battery impact on its own device, the proposition of offloading is more favourable than computing itself. Also, as the *onloader* has no part in making the decision, some of the sensors were even more overloaded than in the case of NO case. This highlights the importance of the weight selection for the decision making.

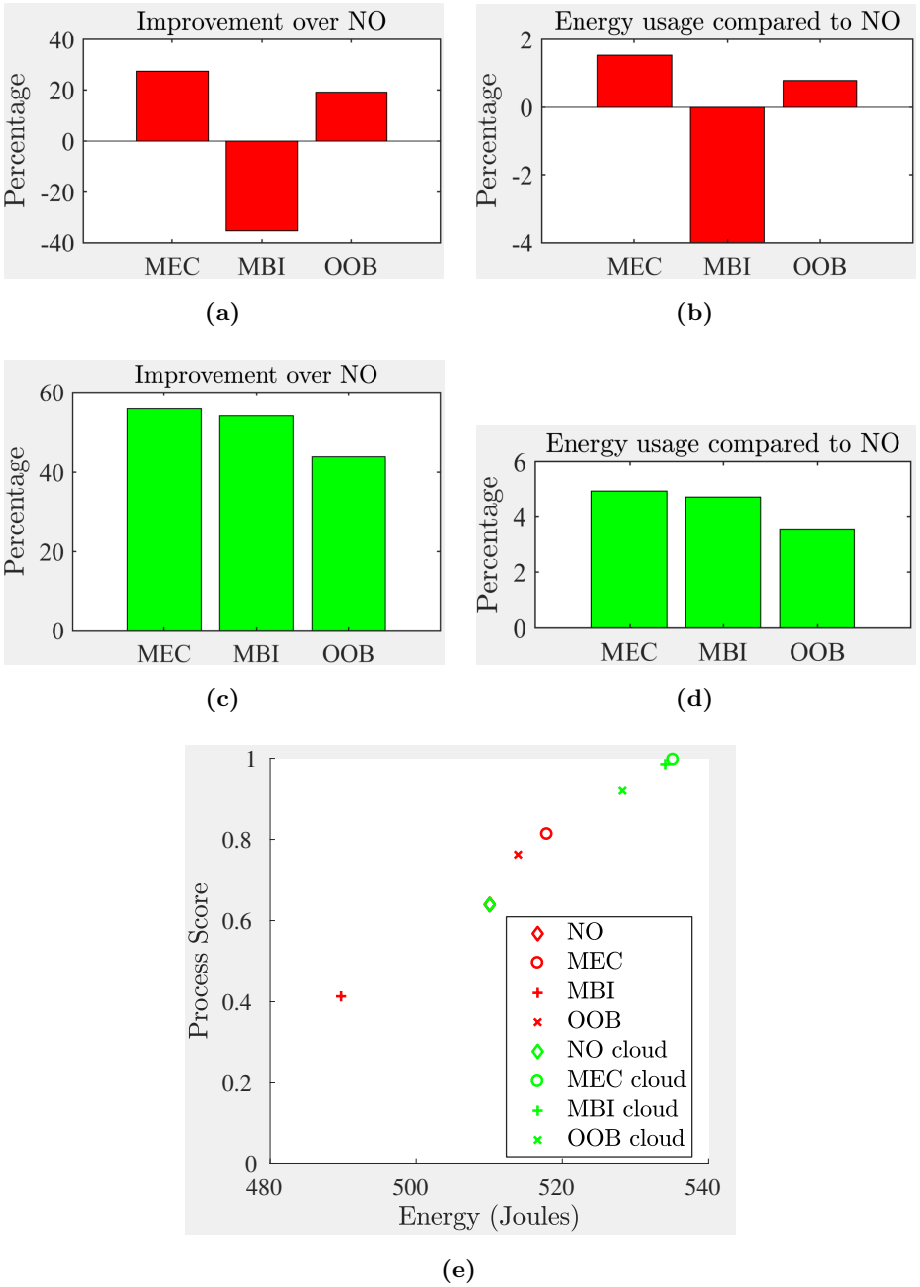


Figure 4.13: Algorithm performance for selfish and Heterogeneous sensors (a) Performance Increment and when cloud is not available. (b) Extra energy usage compared to NO when cloud is not available. (c) Performance Increment when cloud is available. (d) Extra energy usage when cloud is available. (e) Process Score vs Energy usage.

4.7 Conclusion

This chapter presented a real usage of *Computation Offloading* for computationally intensive algorithms to offload to neighbouring devices when the cloud is not available. For that, a simple multi-objective problem is formulated and novel methods are used to find the Pareto-Optimal solution of the problem including the ε constraint method. Based on this, three different algorithms are proposed namely MEC, MBI, OOB.

A holistic simulator is developed to simulate the system and the environment and evaluate the performance of the proposed algorithms. The simulator is simplistic, flexible, and easy to use. Using the simulator, the proposed algorithms are tested extensively in several environments including selfish/unselfish, homogeneous/heterogeneous, and cloud/non-cloud scenarios. The results showed that among the three, MEC consistently achieved the best trade-off between power and performance. It improved the performance by $20 - 40\%$ while costing less than 5% more energy in the non-cloud scenario. When cloud is available, the dropping of targets were reduced by $30 - 55\%$, while the additional energy usage is only about 5% more than that of NO case.

A new metric called ES is formulated that is used to compare the efficiency of the algorithms. In case of OOB, the algorithm is performed slightly lower than the MEC due to the fact that it only offloaded when the nodes could not do the job itself. If the bandwidth is constrained, this could be an advantage over other algorithms. The results also highlighted the importance of weights for the objective function. When the weight for neighbouring devices were set to 0 (selfish case), the overall performance for MBI was far worse than that of NO case.

The results confirmed the hypothesis that *Computation Offloading* could be useful even when the cloud is not available. Even with the availability of the cloud, neighbours can be of significant alternative depending on the network bandwidth. In this chapter, an ideal environment is chosen such that all the nodes can communicate with each other and every node has accurate sensor information about other nodes at all times. This

may not be true in the real world. Some of these shortcomings are explored in the next chapter.

Chapter 5

Computation Offloading based on Queueing Theory

5.1 Introduction

In the last chapter, a novel framework was presented whereby computationally intensive algorithms could be offloaded to neighbouring devices. This offloading was termed as *Computation Offloading* and three algorithms namely Minimal Energy Cost (MEC), Minimal Battery Impact (MBI) and Offload Only if Busy (OOB) were proposed. The proposed algorithms showed considerable performance enhancement compared to the Non Offloading (NO) case with reasonable energy overhead. However, in the simulation, there were many environmental settings, which were deemed to be ideal. For example, all the information about the neighbours were assumed to be known, and it was assumed that each sensor could communicate with any other sensors in the network. Similarly, there were other shortcomings as well, such as, the decision had to be made for every new job and the impact of the proposed algorithm itself were ignored. So, although they were helpful in establishing that *Computation Offloading* can be helpful in spite of the cloud not being available, there were a lot of things to consider for it to be applied in practice.

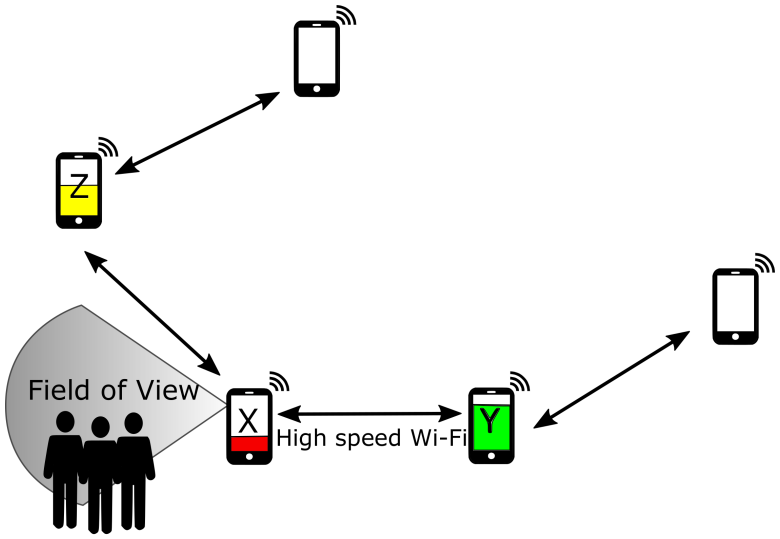


Figure 5.1: Pedestrian identification scenario

In this chapter, the issues mentioned above are addressed. Similar to the last chapter, a network of commercial off-the-shelf (COTS) sensors running Person Re-identification (PRID) algorithms (Figure 4.5) is considered as the exemplary problem – see Figure 5.1. The only difference being, the cases with the cloud are not considered in this chapter. There are many environments in real-life where the cloud is not available at all. As explained earlier, the last chapter proved that offloading to neighbours may be beneficial. The argument about helping neighbours may be valid even 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 the recharge time commences. In case of uneven load, by helping neighbours, the network lifetime (the time when the first node in the network runs out of battery) [147] can be extended.

Even if the cloud is available, it may not be desirable, owing to high cost as well as security issues. For example, the recent rise of Internet Of Thing (IOT) [148, 149] mean many low powered devices have communication capabilities with each other and to the cloud. These devices can range from user's smartphones to household appliances like smart televisions, fridges and security cameras and intelligent locks. Uploading the

data produced by these devices to the cloud for processing may be not only expensive regarding time and money but also prone to security breaches.

This chapter uses Queueing Theory described in Section 2.6 to model the nodes especially the network of queues . It abstracts the scheduling algorithms of the underlying hardware so the system may consist of Central Processing Unit (CPU) nodes or dedicated accelerators such as Graphical Processing Unit (GPU) and Field Programmable Gate Array (FPGA). Working with job rate rather than individual jobs, the need to decide for every task is eliminated. As highlighted earlier, the significant difference to the last chapter is that not all nodes in the system can communicate with all other nodes in the network and information about the nodes are not known at all times. In fact, one of the objectives of this chapter is to determine how frequently the information has to be communicated between the sensor nodes.

The next section presents related works on computation offloading when cloud is not available. Section 5.3 models the node network using a network of queues and formulate the problem along with the Node State Information (NSI). In Section 5.6, the algorithms are proposed. Then Section 5.7 discusses some minor changes to the simulator compared to the last chapter and the target data for testing the proposed algorithm. In Section 5.8, the experiments and the results are presented. Finally, Section 5.9 discusses the results and concludes the chapter.

5.2 Related Works

In this section, closely related works to offloading to neighbouring nodes are discussed. Even though there are a plethora of papers in Mobile Cloud Computing (MCC) to the cloud, not many consider neighbouring nodes as candidates for offloading.

The availability and quality of a communication channel have a huge impact on successful offloading. Cuervo [106] 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. Wu et al. [150] 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 [151] 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 [152,153]. In this approach, communication is between neighbouring nodes connected by either WiFi or Bluetooth. As the WiFi and Bluetooth coverage is limited compared to cellular network coverage, interference may be limited as well.

As the development of embedded devices continues, researchers are keen to exploit it. For example, Lin et al. [154] considered offloading to *coprocessors* and Magurawalage et al. [120] considered offloading to cloudlets along with the *cloud*. Their model is selfish as each user only try to optimise their own performance. Whereas in this work, cost of helping on the *onloader* is also considered. Also, in this work, if the node has a *coprocessor*, then it can be considered as a separate node but with no communication cost.

Recently, Truong-Huu et al. [155] also considered smartphones as offloading candidates. Their main objective 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, do not mention 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.

Similarly, Vilaplana et al. [156] used the Open Jackson network to model the cloud

architecture and estimate their performances such as response times and utilisation of the the system. They considered individual queues for resources such as load balancer ($M/M/1$), processing server ($M/M/m$) and database ($M/M/1$).

5.3 System Model

In this work, sensors and their communication links are modelled as nodes and arcs respectively. Let $G = (S, E)$ be a directed network defined by a set S of $|S|$ nodes and a set E of $|E|$ directed arcs. Each arc $(i, j) \in E$ 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. The link between two nodes is dependent on the proximity of the nodes and can also be multi-hop. For example, Figure 5.2 shows seven nodes with communication links between some of them. For node '5' to communicate with '3' and '6', it has to communicate via node '7'. Based on this network, the cost of communicating to each node to every other node can be calculated using shortest path algorithms defined in Chapter 2. This is similar to route discovery phase in Mobile Ad-hoc Network (MANET) routing protocols Destination-Sequenced Distance-Vector (DSDV) and Ad-hoc On-demand Distance Vector (AODV).

5.3.1 Node

Each node i is a smart camera with with a CPU and communication capabilities. $M/M/1$ queues described in Section 2.6 are used to model the behaviour of each of these components. Figure 5.3 shows a node being modelled using queues. A CPU is represented by a $M/M/1$ queue. Similarly, for the communication part, WiFi is modelled using two $M/M/1$ queues (sending and receiving side). Without any loss of generalisation, common WiFi send and receive rate (i.e $\mu_{iWS} = \mu_{iWR} = \mu_{iWF}$) is assumed. Each node i may be defined as a tuple $\{\gamma_i, \gamma_{0_i}, \mu_i, \mu_{iWF}\}$ where γ_i is the rate of *offloadable jobs*, γ_{0_i} is the rate of *non-offloadable jobs*, μ_{iCPU} is the service rate of

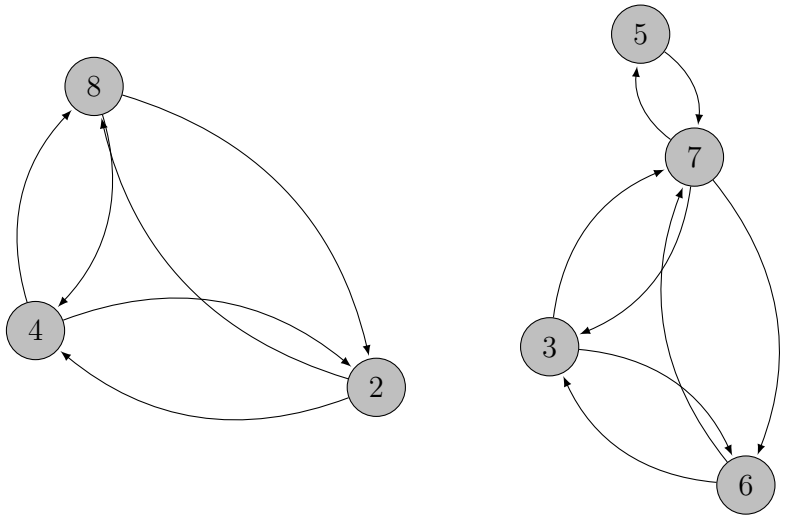


Figure 5.2: Network of sensors highlighting partial connectivity. Each node is modelled as shown in Figure 5.3.

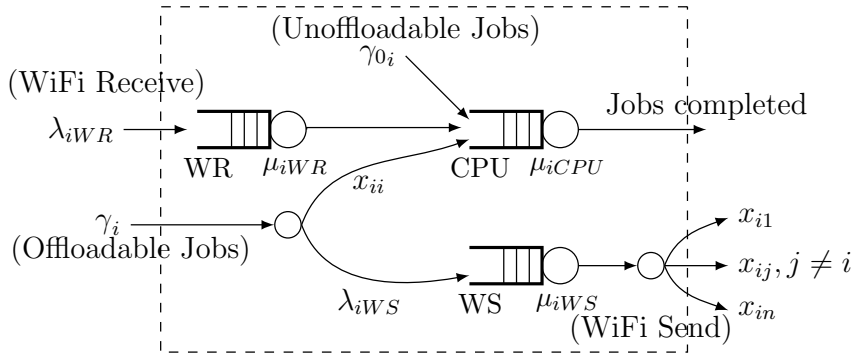


Figure 5.3: A sensor node modelled as network of queues. CPU, WR, WS represent CPU, WiFi Receiver and WiFi Sender queues respectively

CPU and μ_{iWF} is the WiFi transmission rate. This node information is defined as the Node State Information (NSI).

5.3.2 Arcs

The communication cost (time as well as energy) depends upon the Network Bandwidth (BW) between the nodes and data size. However, the communication channel is not perfect due to various noises and interference. The network data rate may be adjusted based on these factors using metrics such as Signal to Noise Ratio (SNR) and acknowledgement. for optimal performance which is to offer high bandwidth at high Packet Delivery Rate (PDR) [74, 157]. Results from [157] show that depending on SNR, the PDR can be different for different data rates. So in order to model their behaviour correctly, this work accounts for them using a retransmission factor r. To understand how and why it is used, say the current channel contains noise and interference. The transmission process is a well defined Bernoulli trial with Bit Error Rate (BER)(β) being the probability of failure for each trial. BER is independent between trials. Let a packet of data contain $\mathcal{D}$ bits of data. Due to the data redundancy, interleaving, in the communication protocols, let the system be able to correct up to e bits of error. That means, while transmitting a packet of data containing $\mathcal{D}$ bits, if number of bits that are corrupted during the transmission of data is less than or equal to e then the packet has been delivered successfully and vice versa. The probability of the packet transmission resulting in success of PDR can be calculated using a Binomial distribution [158].

$$PDR = \sum_{k=0}^{e} \binom{\mathcal{D}}{k} \beta^k (1-\beta)^{\mathcal{D}-k} \tag{5.1}$$

Depending on the number of bits in the packet, number of errors correctable and BER the PDR will vary. Figure 5.4 shows their relationship for a few selected cases. It shows that a longer packet length is more susceptible to error than a shorter packet length; even a small margin for error (ie. $e > 0$) can increase the PDR significantly. In the experiments, instead of sampling the SNR, calculating the BER and in turn estimating the equivalent PDR, PDR between two nodes is directly sampled from a uniform

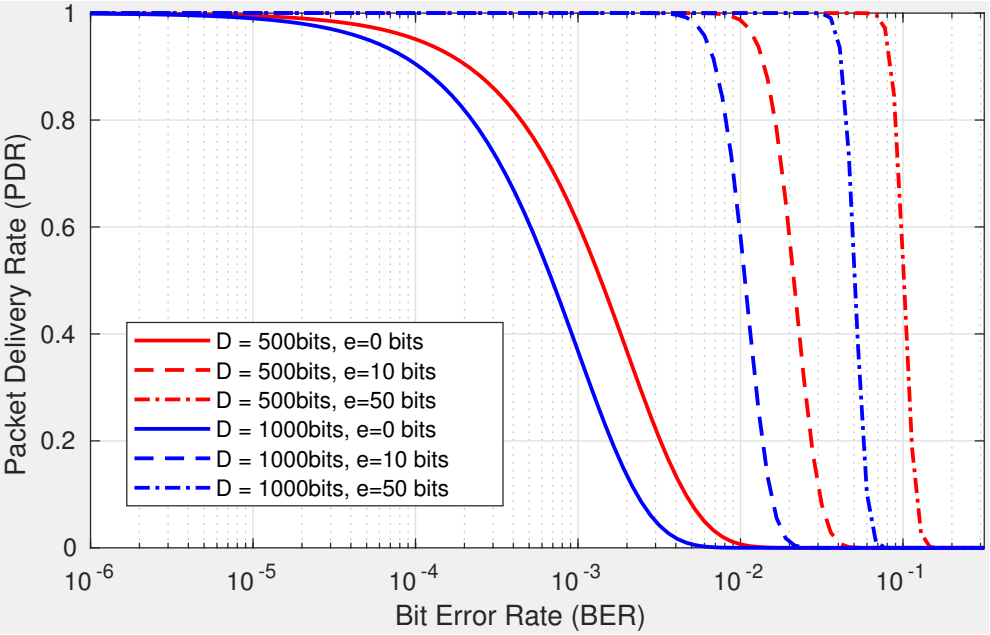


Figure 5.4: PDR in presence of noise and interference.

distribution. A lost or damaged packet of data has to be transmitted again resulting in time and energy loss until it has been successfully received by the receiver. For example, let the PDR of the channel be 0.4. This means the packet will be successfully received by the receiver 40% of the time in the first trial. For the 60% of the time, the sender has to transmit the packet again and the second trial is independent to the first trial. The second trial will also fail 60% of the time and so on. This phenomenon can be modelled using a negative Binomial distribution or a geometric distribution as shown in Figure 5.5. In this case, the mean of the distribution can used to calculate the average number of transmissions to send the data from one node to another:

$$r(PDR) = \mathbb{E}[g(x; PDR)], \text{ where} \tag{5.2a}$$

$$g(x; PDR) = PDR(1 - PDR)^{x-1}, \forall x \in \{0, .., \infty\} \tag{5.2b}$$

The relationship (see Figure 5.6) 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, average number of times the data has to be transmitted is $1, 2$ and 9 times, respectively. In Section 5.6.2, further analysis is performed to see effect of bandwidth, PDR, and the

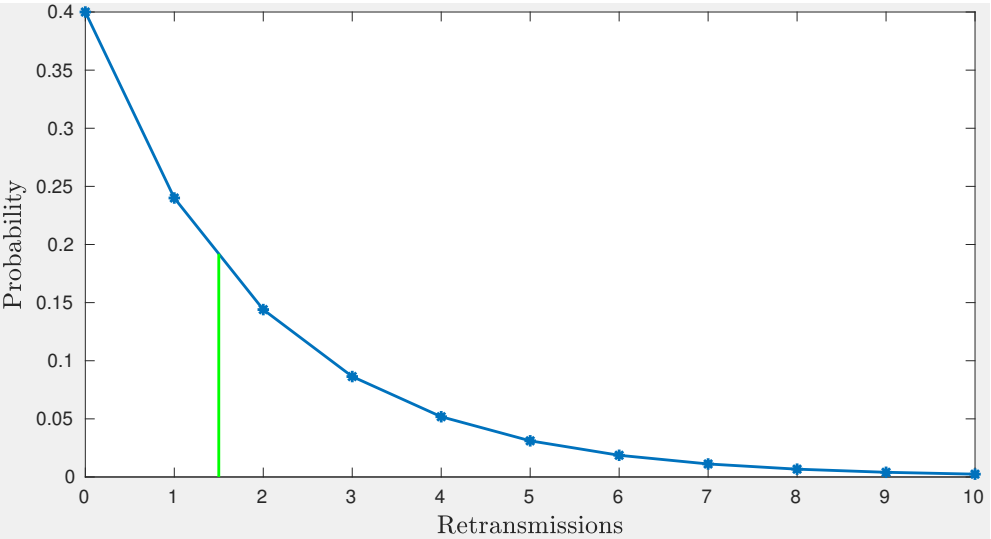


Figure 5.5: Probability distribution of number of retransmission required for $PDR = 0.4$. The green stem at 1.5 is the mean of the distribution.

frequency of NSI exchanges. For the simulations, 0.5 is considered as the minimum PDR for any valid communication link. Once the retransmission factor is known, the communication cost between i and j, α_{ij} is defined as:

$$\alpha_{ij} = L_{iWS}T_{iWS} + \mathcal{D} \times \frac{r+1}{BW_{ij}} + L_{jWR}T_{jWR} \tag{5.3}$$

where, BW_{ij} is the bandwidth between node i and j; D is the data size; r is the average retransmission times (see Equation (5.2a)); α_{ij} is the communication cost; B_j is the remaining energy in node j (Joules), L_{iWS}, L_{jWR} are the number of jobs already in the WiFi send and receive queues of node i and j; T_{iWS}, T_{jWR} are expected WiFi sending and receiving time in i and j. Note that α_{ij} can be interpreted as the suitability of node j based on existing communication queues and the channel available.

5.4 Case Study: Three Nodes

In order to illustrate how the information about the neighbouring nodes can be used in making offloading decision, consider a simple example ignoring effects of the

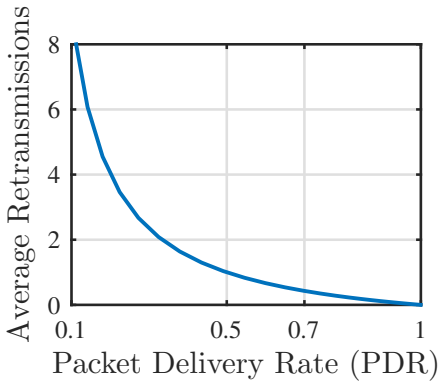


Figure 5.6: Average no. of retransmissions required due to imperfect channel.

communication links. Consider three nodes s_1, s_2 and s_3 each defined by a tuple $\{\gamma, \gamma_0, \mu, \mu_{WF}\}$. Each node is faced with a decision problem to either do it themselves or offload to other nodes. Based on Open Jackson network (Section 2.6.2), the total incoming job rate of the sensor nodes can be expressed as:

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{bmatrix} + \begin{bmatrix} \gamma_{01} \\ \gamma_{02} \\ \gamma_{03} \end{bmatrix} \tag{5.4}$$

Let the serving rates for all nodes be equal to one (i.e. $\mu_1 = \mu_2 = \mu_3 = 1$ jobs per second), and all jobs be offloadable (i.e.$\gamma_{01} = \gamma_{02} = \gamma_{03} = 0$ jobs per second). Let the matrix in Equation (5.4) $\hat{\mathbf{X}}$ be referred to as the policy matrix . It can be shown that using this matrix, the load can be balanced between the nodes. Consider three cases, in the first case, let the external incoming rate γ of all nodes be individually less than their serving rate. The nodes are already rate stable, $\hat{\mathbf{X}}$ can be as chosen to continue without offloading as shown in the first row of Table 5.1.

In the second case, one sensor s_1 is overloaded. In default configuration, s_2 and s_3 are stable and would continue to perform as expected. However, s_1 is unstable, and would continuously miss targets and run out of battery life faster than the rest of the nodes. There are multiple policies that can be used in this case to offload the extra load from s_1. For example, first policy could be to offload jobs that it cannot handle ($\lambda - \mu = 0.4$) to either s_2 or s_3. Another solution would be, as shown in the second row of Table 5.1,

to offload to both s_2 and s_3. Also, instead of offloading only the jobs it cannot process (i.e. $\lambda - \mu$), it offloads so that the load of all three nodes can be as equal as possible. This could offer better response or the sojourn time.

In the third case, two of the nodes are overloaded, but the total rate of external incoming jobs is still less than the total capacity of the system. As shown in the third row of Table 5.1, the solution is to manipulate $\hat{\mathbf{X}}$ such that the overloaded jobs are sent to s_3. An important thing to note here is that the resulting incoming rates for all the nodes is close to the serving rate of the nodes. This would lead to unstable nodes with infinitely growing queue. This is referred as heavy traffic approximation [159, 160]. In the case, when the sum of the incoming job rates is greater than the sum of service rates, a solution that guarantees rate stability of all the queues does not exist. In such scenarios, a solution would be to drop the targets when the queue length is too long.

5.5 Problem Formulation

The above case study illustrates different kinds of problems faced when trying to find the appropriate decision policy. As the number of nodes is generalised to $|S|$, the number of elements in the policy matrix is $|S|^2$. In addition, the cost of communication, the effect of NO jobs, and the heterogeneity of the sensors needs to be accounted for. In Chapter 4, each sensor took its own decision after accessing the status of all the neighbouring nodes. In this work, the problem is formulated as a minimum cost network flow problem. There are centralised and distributed approaches which are described next.

5.5.1 Centralised Problem Formulation

The scheduling decision problem is defined as a minimum cost flow problem to find the optimal policy $\hat{\mathbf{X}}$ 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,

Table 5.1: Different loads possible for three nodes scenario. Green font colour represents stable job rate whereas red font colour represents unstable job rate. When $\Sigma\lambda > \Sigma\mu$, a solution satisfying all nodes is not available

	Case	γ	Default $\boldsymbol{\lambda}, \boldsymbol{\rho}$	Optimised $\hat{\mathbf{X}}$	$\boldsymbol{\lambda}, \boldsymbol{\rho}$	Remarks
i	$\lambda_1 \leq \mu_1$ $\lambda_2 \leq \mu_2$ $\lambda_3 \leq \mu_3$ $\Sigma\lambda \leq \Sigma\mu$	0.3 0.4 0.2	0.3 0.4 0.2	0.3 0 0 0 0.4 0 0 0 0.2	0.3 0.4 0.2	Each incoming rate is less than the service rate.
ii	$\lambda_1 \leq \mu_1$ $\lambda_2 \leq \mu_2$ $\lambda_3 \leq \mu_3$ $\Sigma\lambda \leq \Sigma\mu$	1.4 0.4 0.2	1.4 0.4 0.2	0.7 0.3 0.4 0 0.4 0 0 0 0.2	0.7 0.7 0.6	Sensor s_1 is overloaded. Multiple solutions exist such as offload to minimum nodes or offload to equalise load amongst all nodes (presented here).
iii	$\lambda_1 > \mu_1$ $\lambda_2 > \mu_2$ $\lambda_3 \leq \mu_3$ $\Sigma\lambda \leq \Sigma\mu$	1.3 1.4 0.2	1.3 1.4 0.2	1 0 0.3 0 1 0.4 0 0 0.2	1 1 0.9	Sensors s_1 & s_2 are overloaded, solution is to offload to the s_3.

without compromising the stability of the queues.

$$\hat{\mathbf{X}} = \arg\min_{x} \sum_{i=1}^{|S|} \sum_{j=1}^{|S|} c_{ij} x_{ij} \tag{5.5a}$$

subject to

$$\sum_{j=1}^{|S|} x_{ij} = \gamma_i, \qquad \forall i \in S \tag{5.5b}$$

$$\sum_{j=1}^{|S|} x_{ji} + \gamma_i \preceq \mu_{iCPU}, \qquad \forall i \in S \tag{5.5c}$$

$$x_{ij} \geq 0 \tag{5.5d}$$

The decision variable $x_{ij} \in \mathbb{R}^{(|S|\times|E|)}$ represents the job flow on an communication link $(i,j) \in E$ 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 5.5.3. 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 Equation (5.5b) makes sure that all the jobs are assigned to a processing node whereas the inequality constraint in Equation (5.5c) makes sure that the jobs can be processed by the corresponding nodes they are assigned to. This formulation uses NSI from all the nodes (S) and makes decision for all the nodes simultaneously. Equation (5.5) can be solved using efficient linear programming techniques [129]. The solution of Equation (5.5) can be written as a decision matrix shown below:

$$\hat{\mathbf{X}} = \begin{bmatrix} x_{11} & x_{12} & \dots \\ \vdots & \ddots & \\ x_{|S|1} & & x_{|S||S|} \end{bmatrix} \tag{5.6}$$

Each row of $\hat{\mathbf{X}}$ represents the policy for each node and defined as the policy vector $\hat{\mathbf{x}}$. $\hat{\mathbf{x}}_i$ 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.

5.5.2 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 the policy vector ($\hat{\mathbf{x}}$) 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* [129] whereby each node calculates its own policy vector ($\hat{\mathbf{x}}$). The distributed formulation can then be defined for each node $i \in N$ as shown in

Equation (5.7).

$$\hat{\mathbf{x}}_i = \arg\min_{x} \sum_{j=1}^{|S|} c_{ij} x_{ij} \tag{5.7a}$$

subject to

$$\sum_{j=1}^{|S|} x_{ij} = \gamma_i \tag{5.7b}$$

$$\sum_{i=1}^{|S|} x_{ji} + \gamma_{0i} \preceq \mu_{iCPU} \tag{5.7c}$$

$$x_{ij} \geq 0 \tag{5.7d}$$

This is similar to the *Gauss-Siedel* like method used by Meskar [161] for MCC. The algorithm basically communicates with its immediate neighbours to see what they can offer 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 Equation (5.5) where each node i only tries to minimise the cost of its own objective function on the basis of information available on its neighbours. Similar to centralised problem, Equation (5.7) can also be solved using linear programming techniques [129].

5.5.3 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 Equations (5.5) and (5.7) 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:

$$c_{ij} = \begin{cases} \omega_1 L_i T_i, & \text{if } i = j \\ \omega_1 L_j T_j + \omega_2 \alpha_{ij} + \omega_3 \frac{1}{B_j}, & \text{if } i \neq j, (i,j) \in E \\ \infty, & \text{if } i \neq j, (i,j) \notin E \end{cases} \tag{5.8}$$

where, L_i is the number of CPU jobs already in node i, T_i is the average processing time of each CPU Jobs, α_{ij} is the communication cost defined in Section 5.3.2 and $\{\omega_k\}_1^3$ are weight factors. The significance of various components in Equation (5.8) can be changed using the weighting factor $\{\omega_k\}_1^3$. The algorithms can put more emphasis on one component than the other. For example, if the nodes are mains powered, ω_3 can be set to zero.

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 availabity for further external jobs and vice versa. This is also applicable for self-processing in the scheduling decision making.

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 Equation (5.8) 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. The detailed models of power drain for the CPU, Image sensor and WiFi communications and various energy saving modes were described in the last chapter.

5.5.4 Computational Complexity

The main objective of this work is to balance the workload of the nodes in energy limited scenario by offloading computationally intensive algorithms. It will be inefficient if the proposed algorithms use a significant amount of CPU and communication resources to

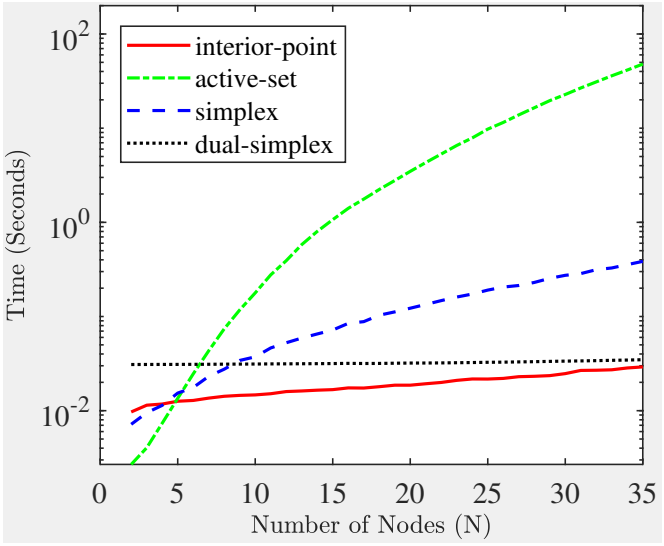


Figure 5.7: Time complexity of various linear solvers

balance the computational load. The optimisation problem stated in Equations (5.5) and (5.7) can be solved using efficient linear programming techniques. One advantage of using data rate in the problem formulation is that there is no integer constraint. 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. Experiments were carried out to gauge their *time complexity* for a different number of nodes and found the interior point to be the most efficient – see Figure 5.7. 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. However, solving this optimisation problem in one of the nodes periodically would have a negative impact on its battery life. Also considering an ad-hoc nature, cluster-heads which are in charge of the scheduler are not defined. Instead, two simple approaches are used. First is to run the scheduler in a round-robin fashion among the nodes (see Section 5.6.2). Second is to run them in a distributed fashion so that each node has an optimisation problem to solve, in order to find the optimal policy (see Sections 5.6.3 and 5.6.4). In both approaches, the NSI data has to be circulated amongst the neighbours so that the problem can be solved. This would be discussed in detail later.

Table 5.2: Algorithms proposed in this work.

Algorithm	Centralised	Pro-active
Oracle (O)	✓	✓
Proactive Centralised (PC)	✓	✓
Proactive Distributed (PD)		✓
Reactive Distributed (RD)		

5.6 Algorithms

In Sections 5.5.1 and 5.5.2, 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 –see Table 5.2. 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 Equations (5.5) and (5.7)). However, an assumption is made that the nodes are not selfish and only offloads if required.

5.6.1 Oracle

The target detection rate varies with time so the job rates (γ) in Equations (5.5) and (5.7) are non-stationary. The lowest sampling time of the simulator is $10ms$, hence the problems in Equations (5.5) and (5.7) 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 Equation (5.5) every second which is every hundredth sampling step. Once solved, it sends the related policy $\hat{\mathbf{x}}_i$ to all nodes simultaneously without using the communication

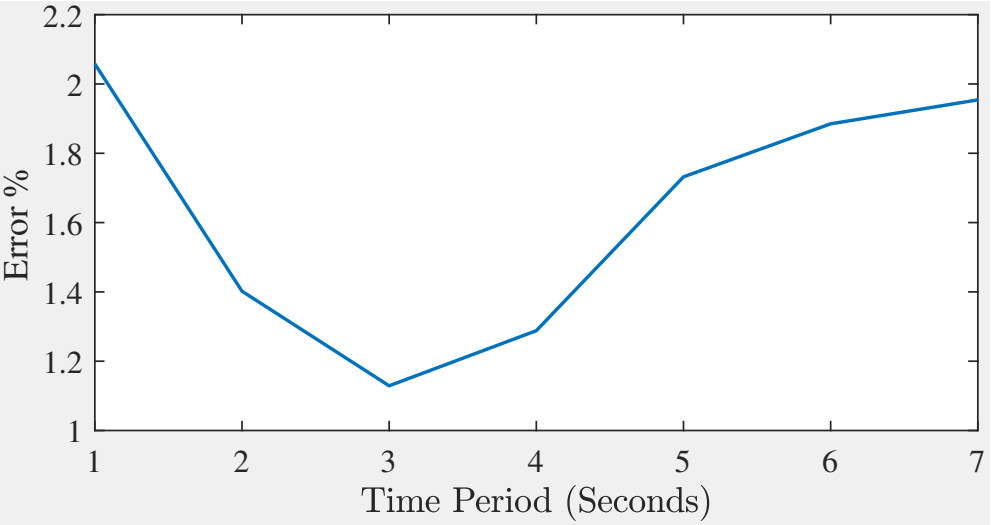


Figure 5.8: The number of tasks dropped increases as the duration is too short. Similarly when the duration is long, number of task dropped are increases as well. The best results are achieved at 3 seconds

channel. While this continued update of NSI, is not feasible in practice, it provides a benchmark for comparison. Also, NSI contains the average values of CPU load, incoming rates and service rates. In order to find the best duration to calculate the averages, a simple experiment is conducted. In this experiment, Oracle is run multiple time in the same setting except for the duration of NSI calculation which is variable. When the average is based on the last second of activity, the around two percent of targets were dropped. As this duration increased, the performance is best at three seconds. After that further increment in the duration worsen the performance – see Figure 5.8.

5.6.2 Proactive Centralised

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 $(|S| - 1)$ nodes send their NSI to it periodically. Similar to Oracle, the server solves Equation (5.5) and sends the corresponding policy $\hat{\mathbf{x}}$ back to each nodes. All other nodes are obliged to follow the decision made by the *server* and computes and offloads based on the policy $\hat{\mathbf{x}}_i$ until a

new one is broadcast. However, different to Oracle, the cost of communication, as well as cost of executing the solver are taken into account.

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 on a *round-robin* basis. A different *server* is chosen which acts as the *server* based on time trigger.

There arises a question, 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 $|S|$ nodes in total, and $|S|-1$ nodes sending their NSI to the *server* every t seconds, the node with the highest probability of being busy is the nominated *server*. The arrival rate, worst service rate and the utilisation of the *server*'s receiving queue can be calculated as follows:

$$\text{Arriving rate }, \lambda = \frac{|S|-1}{t} \tag{5.9}$$

$$\text{Worst Service rate }, \mu = \frac{\text{Data Rate} \times \text{worst PDR}}{\text{NSI size}} \tag{5.10}$$

$$\text{Utilization }, \rho = \frac{\lambda}{\mu} = \frac{(|S|-1) \times \text{ NSI size}}{t \times \text{Data Rate} \times PDR} \tag{5.11}$$

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

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

Based on the arriving rate and service rate, 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 Equation (5.2a) 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. Figure 5.9 shows waiting times at the receiving node at various intervals and for different speeds.

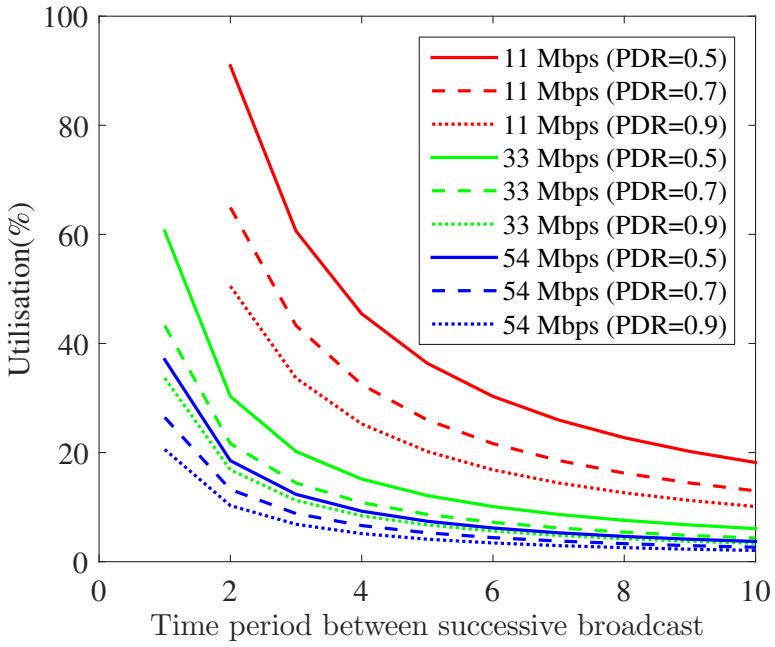


Figure 5.9: Queue utilisation of *server* in proactive setting under various network conditions (Lower is better). Data size set at 1 Mb.

For the data rate of 11 Mbps (red lines in Figure 5.9) 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.

5.6.3 Proactive Distributed

Proactive Distributed (PD) is similar to Proactive Centralised (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 Equation (5.5), each node only solves distributed problem in Equation (5.7).

3. Set S contains immediate rather than neighbours than all the nodes. Even if

total nodes is large (> 100), S may be limited to tens of nodes. For example, see Figure 5.10b, node 1 and 5 are only connected to one another.

This method and the PC could be aligned with the proactive network routing protocols defined in Section 2.5.2 and the NSI can be incorporated in the network update packets.

5.6.4 Reactive Distributed

If a few nodes become overloaded infrequently, transmitting NSI regularly can be a waste of energy. Also, the tail-end behaviour of the User Equipment (UE) (see Section 4.5.1) may mean regular transmission forces UE to stay in the high powered state instead of the low powered idle state [140]. In this method (see Algorithm 4), nodes only communicate when they need to offload. This could fit with the reactive network routing protocols such as AODV and Dynamic Source Routing (DSR) described in Section 2.5.2. In this method, 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 Equation (5.7). 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 Request For Help (RFH).

5.7 Simulator

This chapter uses the simulator described in Section 4.5. One difference, however, is that in the communication channel, the noise and interference are taken into account which were ignored for simplicity in the last chapter. This was explained in detail in Section 5.3.2. In the simulator, this is accounted for increasing the data size based on the channel condition. In simple words, if the channel has the PDR of 0.5 then, the

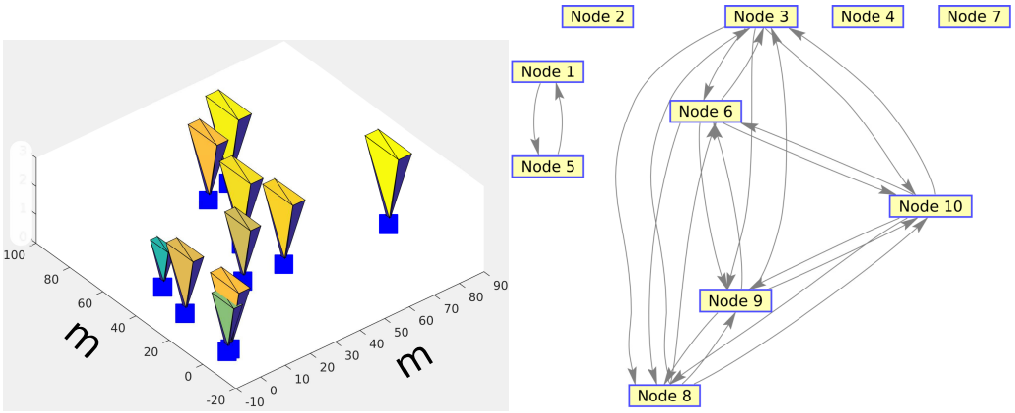
Algorithm 4: Reactive Distributed

Input: Offloadable job rate (γ), Non offloadable job rate (γ_0), Node State Information (NSI)Service rate (μ)

Output: *decision vector*$(dv)_i$

Parameter : Threshold Time ε

```
1  if γ_i + γ_0i ≤ μ_i then
2      Set dv_i to not offload.
3  else
4      if RFH broadcasted & decision time< ε_th then
5          Follow previous dv_i
6      else
7          Broadcast RFH to all nodes.
8          Wait T_wait seconds for NSI
9          if No of NSI received ≥ 2 then
10             Solve Eqn.(5.7) for new dv_i and follow it.
11         else
12             Broadcast RFH again, follow previous dv_i.
13         end
14     end
15 end
```



(a) (b)

Figure 5.10: Simulation setup for one monte-carlo simulation.

simulator assumes that it will take the time and energy of sending two packets, to send a packet to the receiver.

The algorithms are tested on two different datasets. The first uses the Random Waypoint Model (RWP) similar to the last chapter, and the second uses real data from a computer vision dataset. They are briefly described below.

5.7.1 Random Waypoint Model

The RWP model used here is similar to the last chapter except for the consideration of the three dimensional platform. In the last chapter, the platform was considered to be two dimensional for simplicity, and the targets only moved on a plane. However, in real life, things move in three dimension, especially the airborne targets can have six degrees of freedom and move in all three directions. In this chapter, targets move in three dimensions for the RWP (Section 2.5.3) so in order to capture the dynamics, the Field Of View (FOV) of the camera sensors are three dimensional as well as shown in Figure 5.10a.

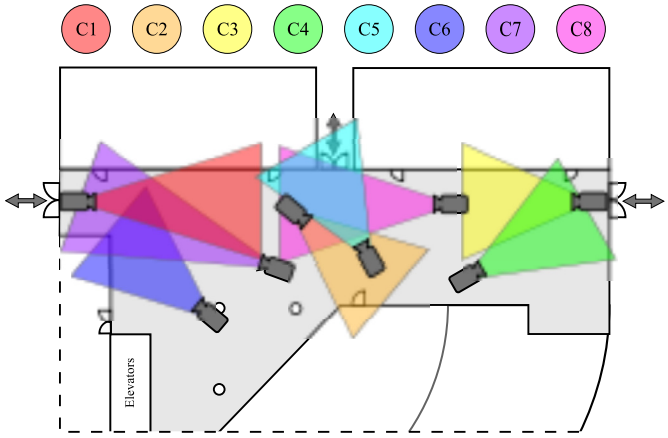


Figure 5.11: Camera placement of SAIVT dataset [162]

5.7.2 SAIVT

A multi-camera scenario described in SAIVT Multi-Camera Surveillance Database [162] 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 system consists of eight cameras placed as depicted in Fig. (5.11). The target tracks for the simulator were extracted from the Extensible Markup Language (XML) files provided with the dataset instead of processing the images. According to the dataset [162], the acquisition rate was 25 Frames Per Second (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.(5.12). The majority of targets are detected by Cameras 1, 7 and 15.

5.8 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 5.7. The total energy consumption for each sensor was estimated by summing power consumption of each component based on energy values from Equations (4.18),

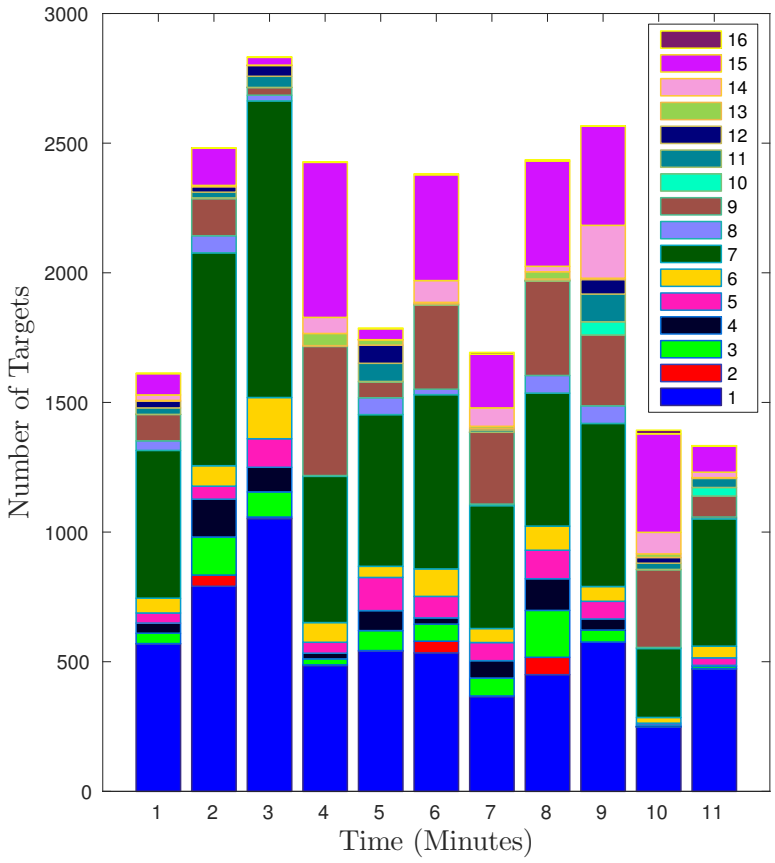


Figure 5.12: Heterogeneous loading of cameras in multi-camera scenario. Camera $1, 7$ and 15 see majority of targets. (Best viewed in colour)

Table 5.3: 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	$30, 60, 90$

(4.19) and (4.22) in Section 4.5. For each run, the simulator was initialised as per Algorithm 5. Each simulation was repeated for the various parameters to see if there is any effect on algorithm performance – see Table 5.3.

Algorithm 5: Simulator initialisation

Input: Sensor set (S)

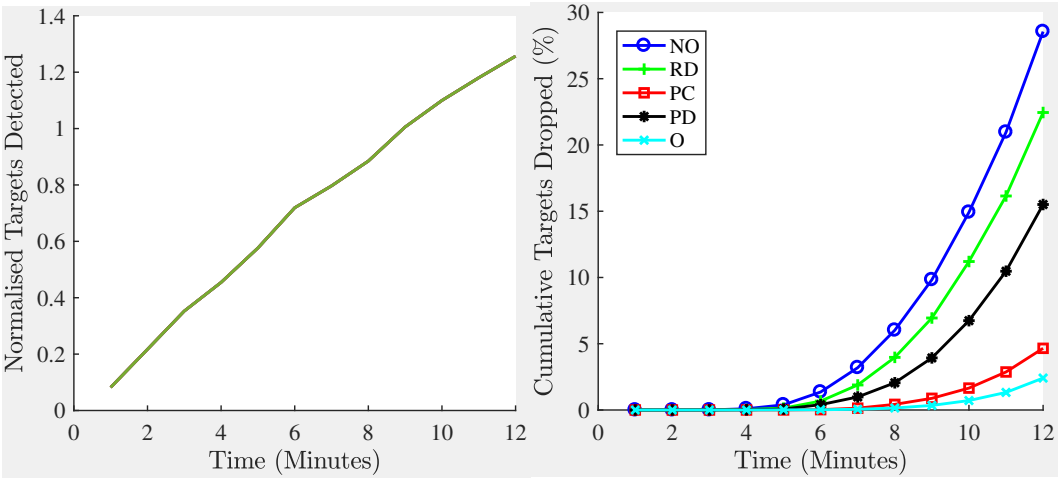
Parameter: Communication range

Output: Location of sensors, Cost of communication between sensors

```
for i ∈ S do
    Generate location of i^th sensor randomly on the platform.
end
for i ∈ S do
    for j ∈ S do
        if Distance (i,j) within communication range then
            Add communication links (i,j) to E
            For each link, randomly generate PDR described in Section 5.3.2
            Use shortest path algorithm to calculate cost per bit between nodes.
            /* The cost can range between 0 (ie same node) to ∞ (i.e.
               no communication link).                                  */
        end
    end
end
```

5.8.1 Results for the Standard Configuration

Figure 5.13a 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 5.3. 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. Figure 5.13b shows the results for the standard configuration of 11 Mbps, communication range of 60m and NSI exchange every five seconds. In the baseline NO case, about 30% of all targets are dropped. The Reactive Distributed (RD) does slightly better than the NO and drops only about 25%. The PD however, performs quite well and

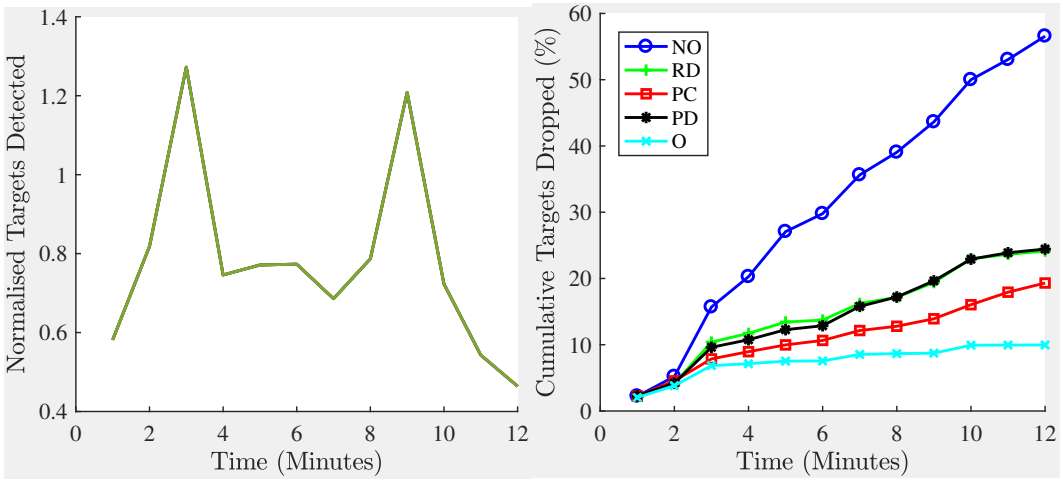


(a) (b)

Figure 5.13: 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.

drops approximately 40% fewer targets. The performance of centralised algorithms is significantly better. The PC and the Oracle (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 a later section.

Fig. 5.14a 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



(a) (b)

Figure 5.14: 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.

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 Figure 5.14a. In the remaining sub-sections, the performance is analysed with respect to the energy consumed as well as effect of environment and parameter selections.

The overall result is summarised in Table 5.4 and Figure 5.15 for the standard configuration. For Figure 5.15, 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 Efficiency Score (ES) metric gives an insight if the extra energy consumed is justified and can help in selecting the right algorithm. This can be explained using an example, in Figure 5.15a, 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

Table 5.4: 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	Algorithm	Arrival Rate (/min)	Service Rate (/min)	Process Score	Energy Used (Joules)	Efficiency Score (Ident/J)
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

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 observed in Figure 5.15 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 5.8.3). 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.

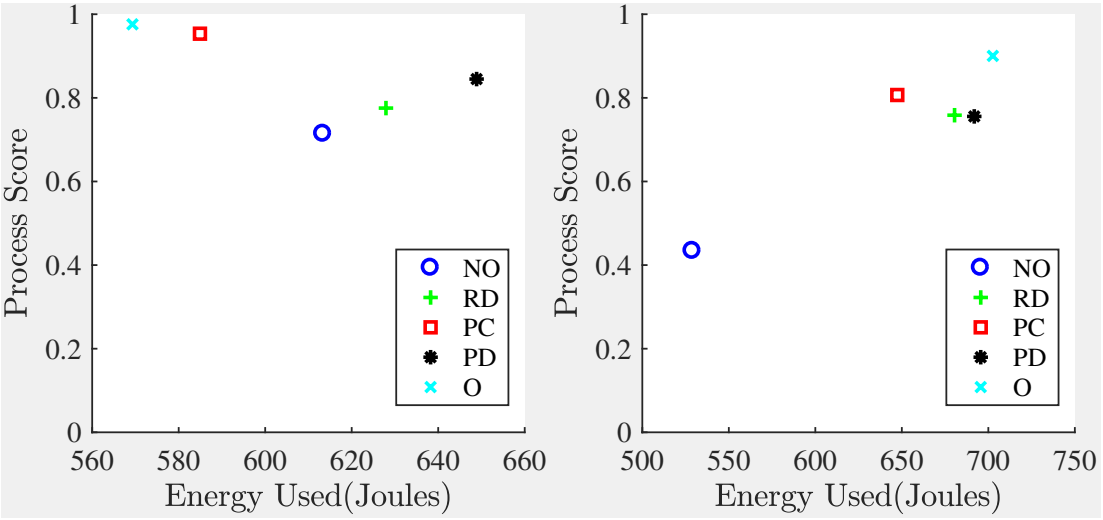


Figure 5.15: Efficiency Scores (a) RWP (b) SAIVT

5.8.2 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 Figure 5.16a. 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 Figure 5.16b). 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 Equation (4.22), 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.

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

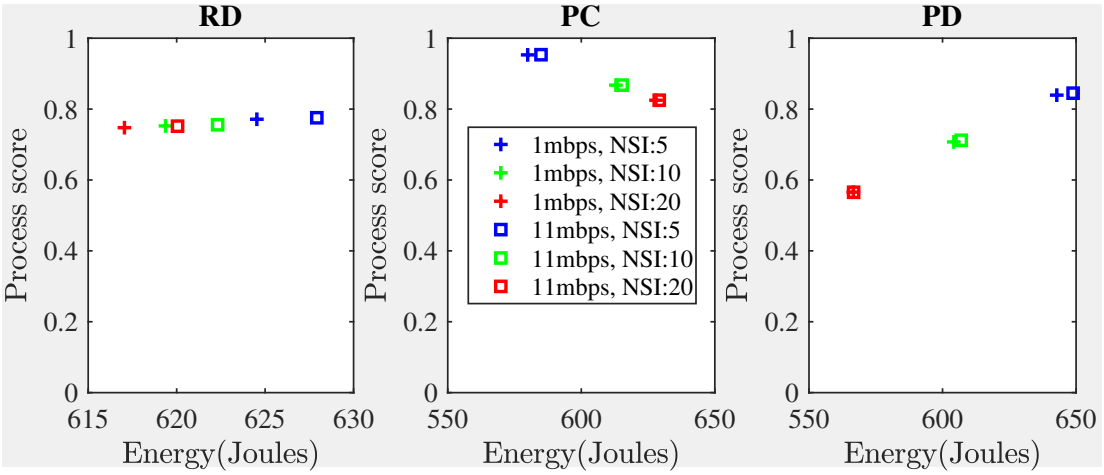
seconds), it performed worse than the NO case, but when it was higher, the performance was better. The trend was consistent in both target datasets. For RD the NSI frequency rate should have no effect because it is asynchronous and nodes communicates with its neighbours when they seek help only. However, as seen in Figure 5.16, 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 – see Fig. Figure 5.16a. However, upon further study, the energy usage was based more on CPU usage than on NSI exchanges.

5.8.3 Effect of Communication Range

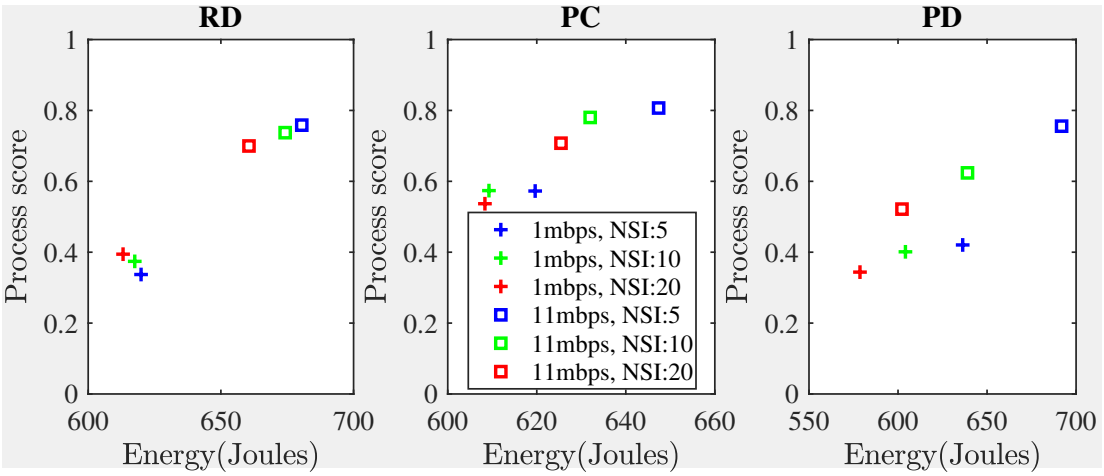
As the communication range of a node is increased, the number of neighbours the node can talk to increases (and vice-versa) – see Algorithm 5. The range was changed to see how the algorithms behave in varying conditions. Heuristically, more neighbours mean more options so the proposed algorithms should perform better when the communication range increases and vice-versa. The experiments generally follow this belief and the results are shown in Figure 5.17. 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.

5.8.4 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. 5.18 shows the average spread of CPU utilisation among the nodes. For RWP, the median



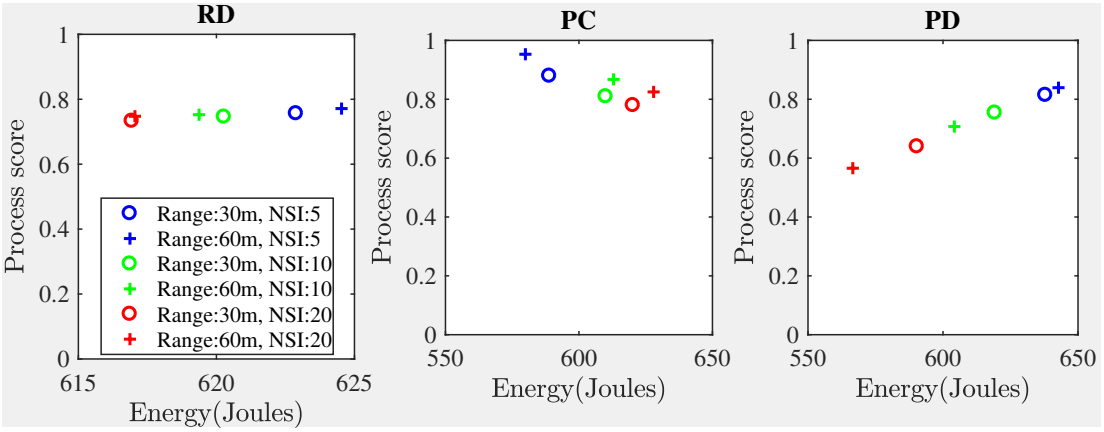
(a)



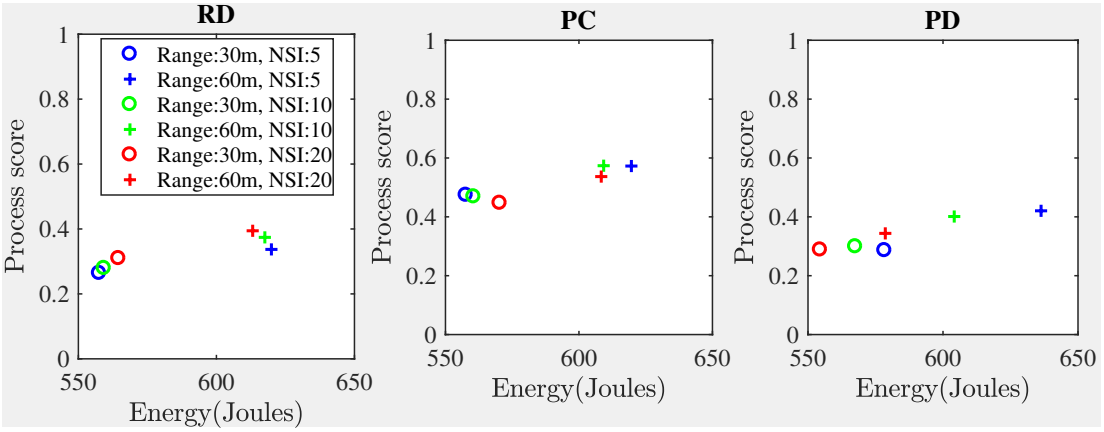
(b)

Figure 5.16: 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.

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.



(a)

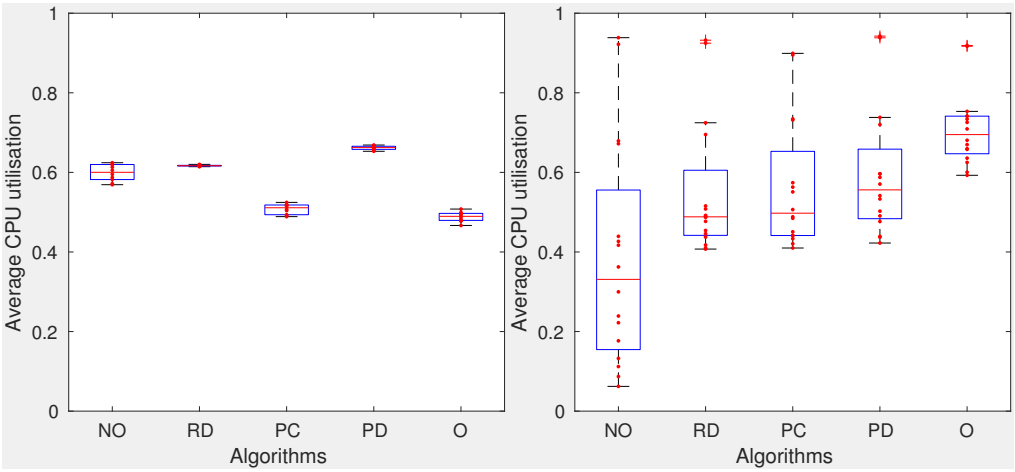


(b)

Figure 5.17: 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**

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 Figure 5.18b). 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



(a) **(b)**

Figure 5.18: Average CPU utilisation across the nodes and NSI frequency (5, 10, and 30 seconds) (a) RWP (b) SAIVT

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.

5.8.5 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 5.5. 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 case and often better. The O had the shortest execution time of all the algorithms tested including the baseline for the RWP dataset,

Table 5.5: 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

whereas PC had the shortest time for the SAIVT case. The centralised algorithms performed better in this metric which may be due to the fact that it is able to consider all the neighbouring states and less likely to make wrong assumptions about neighbours.

5.9 Conclusion

In this chapter, the sensor network is modelled as a network of queues using an Open Jackson network model. In comparison to the last chapter, there are significant improvements and enhancements to the algorithms and simulation. First of all, the network is dynamic and includes noise and interference. Even though the underlying objective function in Chapter 4 is simple, the decision is taken in ad-hoc manner for each job as they arrive and each decision making involves exhaustive search of all the nodes. So, it still is resource intensive and requires significant communication bandwidth. The MEC and MBI algorithms proposed in Section 4.4 are similar to the RD solution in this chapter except for the cost function and the partial connectivity of the network.

In this chapter, various novel *reactive* and *proactive* algorithms are proposed. These algorithms are comprehensively tested in different environmental settings for two different target dataset namely, Random Waypoint Model and a real SAIVT person re-identification dataset. The results show that they significantly enhanced the performance of the system compared to the Non Offloading scenario. Even though, a target identification problem was chosen, the result can be generalised to other problem with computationally intensive algorithms. Also, 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 other nodes NSI are available. Especially in the real dataset, the

performance improvements are significant. The performance boost also comes at similar energy cost and may well increase the network lifetime.

From the experimental results, PC is the best among the three proposed as it has better work completion rate even in the lower communication BW and fewer updates of NSI. Similarly, RD is better if the nodes in the network keeps changing its position such that the existing communication links are broken and new links are formed rapidly. The *proactive* and *reactive* algorithms can also be selected based on the target and sensor activity. For example, if the target activity is high, proactive can be selected and if the target activity is low, reactive approach can be selected. This can also be selected based on the network routing protocol that is being used in the system. For example, if the network is using reactive network routing protocol, selecting RD could be beneficial as the NSI broadcasting and RFH messages can be added unto network update messages and route discovery messages. Likewise, if the proactive network routing protocols are selected, NSI can be circulated along with the network update messages.

Chapter 6

Conclusion and Future Work

This work addressed the feasibility of *Computation Offloading* for real time execution of computationally intensive algorithms on network of embedded devices and proposed novel algorithms that can profoundly enhance the work completion rate of the system even when the cloud is not available. The research carried out in this project is a step towards how systems can be made more efficient in deployment scenarios. The algorithms proposed in this work are not limited to computer vision algorithms but any computationally intensive algorithm that can benefit from offloading. The proposed algorithms are useful in making edge devices self sufficient and autonomous. As such, it can find its usage in the various scenarios where the cloud is not available at all or in commercial Internet of Things (IOT) applications where the focus is more on lowering the demand on the cloud. In addition, there are security concerns while using the cloud as the use is not in control where their data resides and who the data is shared amongst. While taking help from neighbours, some of these concerns can be mitigated. For example, the data shared amongst the neighbours are in the same geographical vicinity and could belong to the same individual or organisation.

The first contribution of this thesis is the comprehensive study of Person Re-identification (PRID) algorithms. The experiment conducted in the chapter quantify the time and energy requirements of recent PRID algorithms when executed on a resource limited

device and the cost of sending data on the network for computation offloading. Based on time taken and data size requirement, Keep It Simple and Straightforward MEtric (KISSME) is the best amongst three compared algorithms to be implemented on an embedded device. These results are taken into account while designing the holistic simulator presented in Chapter 4 and cost functions used in Chapters 4 and 5.

The second contribution of this work is the multi-objective optimisation framework for offloading computations in the network which formulates of the computational offloading problem as a multi-objective problem for with Pareto optimal solutions can be obtained. Also a holistic simulator for evaluating the performance of computation offloading algorithms is presented. Extensive results showing how the algorithms can perform in various scenarios. Based on these experiments and results, Minimal Energy Cost (MEC) is the best algorithm. However, Offload Only if Busy (OOB) is not far behind in terms of performance while not offloading as many as in the case of MEC. If the bandwidth between the sensors are further constrained, the OOB is a better alternative. A novel metric is proposed that is simple to use and intuitive to evaluate the efficiency of algorithms. The simulator and the metric can be used beyond the scope of this work as well.

The major contribution of this project is the modelling of the network in a more realistic fashion using Queueing Theory . The model considers noise and interference on wireless links as well as the partial connectivity between sensors. New centralised and distributed algorithms are proposed, that can significantly improve the work completion rate of the system when the cloud is not available. When the network is dynamic such that the nodes are arriving and leaving at the same time, it is efficient to choose the reactive Reactive Distributed (RD) algorithm. It doesn't require Node State Information (NSI) to be circulated periodically and works pretty well given the Network Bandwidth (BW) conditions are adequate. One such case could be the network of vehicles or Vehicle Ad-hoc Network (VANET). However, if the network is fairly stable, Proactive Centralised (PC) should be chosen whenever possible as it can perform well even when the BW is lower. Proactive Distributed (PD) does not have any significant benefit in terms of performance with respect to the PC but it doesn't require any central scheduler. So in

cases where a central scheduler is hard to assign, PD may be chosen. The results were also evaluated for real computer vision dataset. This evaluation showed considerable improvement in the performance of the system.

6.1 Future Works

Computation Offloading is an interesting subject, particularly in the context of IOT devices and 5G networks. There are many interesting ideas and experiments and works that have been left for the future as it wasn't possible to complete due to the time limitation. Below are the key ideas that should be explored.

In the current state, all the neighbours whose average Central Processing Unit (CPU) load is lower than the threshold should respond to the Request For Help (RFH) message as soon as possible. This could result in higher time and energy cost for the decision maker as well as high probability of collision, particularly if there are too many neighbours. An alternative approach could be to use the response time as the quality of service that the neighbour can offer. In this approach, a neighbour can vary the response time based on how busy they are; meaning they send a quick response when they are idle and have plenty of resources to offer and send a delayed response if they are busy or have limited resources left. The node asking for help can then choose the node which responds quickest or select from the nodes that respond quick enough for their application.

The benefits of the offloading directly depends on quality of NSI. This was discussed in Section 5.8.2. The state of the neighbours can change from when it sends the NSI to when that information is used. However, it is not always possible to increase the NSI frequency to obtain the most up to date NSI. Instead, more robust algorithms could be developed. For example, the system can be modelled using Partially Observed Markov Decision Processes (POMDP) in which the neighbours can be the part of environment. In POMDP, the model is aware that the state of the neighbours can be different than the observed value and act accordingly. The computational complexity of using POMDP can be high for training purposes. However, the policies could be learnt offline.

The simulator has the capability to simulate the movement of sensors, however, this was not explored in this work. In future, this could be explored to see how the effect of movement will effect the performance. If there is significant movement, the nodes in the network would be changing continuously which could mean the proactive algorithm PC has to increase the frequency of NSI updates. This as discussed earlier may not be feasible.

This project explored *Computation Offloading* in the context where all the devices share a common objective. In this work, identifying as many people in the field as possible is the goal of the system. The evaluation of system is accomplished with this in mind. However, in the commercial domain where each sensor accounts for themselves, the evaluation criteria may be changed. The selfishness of the sensors which was briefly explored in Chapter 4 can be explored further. Game Theoretic approach could be applied to analyse and optimise these selfish network of sensors.

Finally, in the current state, the *Computation Offloading* is only based on the cost function relating to the execution of the algorithm and communication between the sensor. However, there may be other factors which could be considered. This was briefly discussed in Section 4.2.4. For example, if in the pedestrian tracking application, if a person is moving from camera '1' to '2', it may be better to offload the *Computation Offloading* job to '2' rather than others, as camera '2' may want to know about the target coming towards it. It would be interesting to see how this could be taken into consideration and possibly exploited.

Appendices

Appendix A

Parameters for Simulation in Chapter 4 and Chapter 5

Table A.1: Parameters used in the Chapter 4 and Chapter 5 simulation

Parameter	Value (Unitless unless specified)	Equation	Remarks
Image Sensor Model		4.18	
P_{idle}	225.4 Joules		
P_{active}	338.8 Joules		
Number of Pixels	800×600 pixels		
Camera Clock Frequency	32 MHz		
T_{frame}	$\frac{1}{\text{Frames per second}}$		
T_{active}	$\frac{\text{Number of Pixels}}{\text{Camera Clock Frequency}}$		
T_{idle}	$T_{\text{frame}} - T_{\text{active}}$		
Cellular Model		4.21	
β_{IDLE}	63.9		
β_{FACH}	267.9		
β_{DCH}	519.3		
Wi-Fi Model		4.22	
β_{LT}	1.2		
β_{HT}	0.8		
$\beta_{\text{LT base}}$	238.7		
$\beta_{\text{HT base}}$	247.0		
Threshold	20 packets per second		
Time costs		9	
Packeting Operations	400 per packet		
T_{packet}	$\frac{\text{Packeting Operations}}{\text{Device Clock Frequency}}$		
T_{send}	$\frac{\text{Send Data Size}}{\text{BW}}$		BW is different for cellular and Wi-Fi
T_{receive}	$\frac{\text{Recieve Data Size}}{\text{BW}}$		BW is different for cellular and Wi-Fi
Cost Function (MEC,MBI, OOB)		4.4	
w_{off}	1		
w_{on}	1		If onloader is a smartphone.
w_{on}	0		If onloader is the Cloud.
w_{time}	1		

Parameter	**Value (Unitless unless specified)**	**Equation**	**Remarks**
Random Waypoint Model		2.5.3	For target simulation
V_{max}	0.6 metre per second		
V_{min}	0.3 metre per second		
Walk Interval	$30 - 400$ seconds		
Pause Interval	$0 - 100$ seconds		
Direction	± 60 deg		
CPU		4.5.1	
Frequency	$\beta_{\mathbf{freq}}$ $\beta_{\mathbf{idle}}$		
245.0	201.0 35.1		
384.0	257.2 39.5		
460.8	286.0 35.2		
499.2	303.7 36.5		
576.0	332.7 39.5		
614.4	356.3 38.5		
652.8	378.4 36.7		
691.2	400.3 39.6		
768.0	443.4 40.2		
806.4	470.7 38.4		
844.8	493.1 43.5		
998.4	559.5 45.6		
Miscellaneous			
τ	5 *seconds*	7	

Appendix B

Simulator Implementation

In this chapter, a brief explanation of how the simulator is implemented. The simulator is made up of four classes described below.

B.1 Sensor

The Sensor class defines the attributes and behaviour of sensor node in the simulator. Its class diagram is depicted in Figure B.1 with its main attributes. It has modules such as CPU, 3G, WiFi, and Camera as described in Section 4.5. Each sensor has a unique identification number (id). A method called "step()" is executed after every sample time, that simulates the running of the sensor for the sample time.

B.2 Target

The target class define the attributes and behaviour of a target. In the simulator, there are many targets at a time moving around in the platform. Its class diagram is shown in Figure B.2

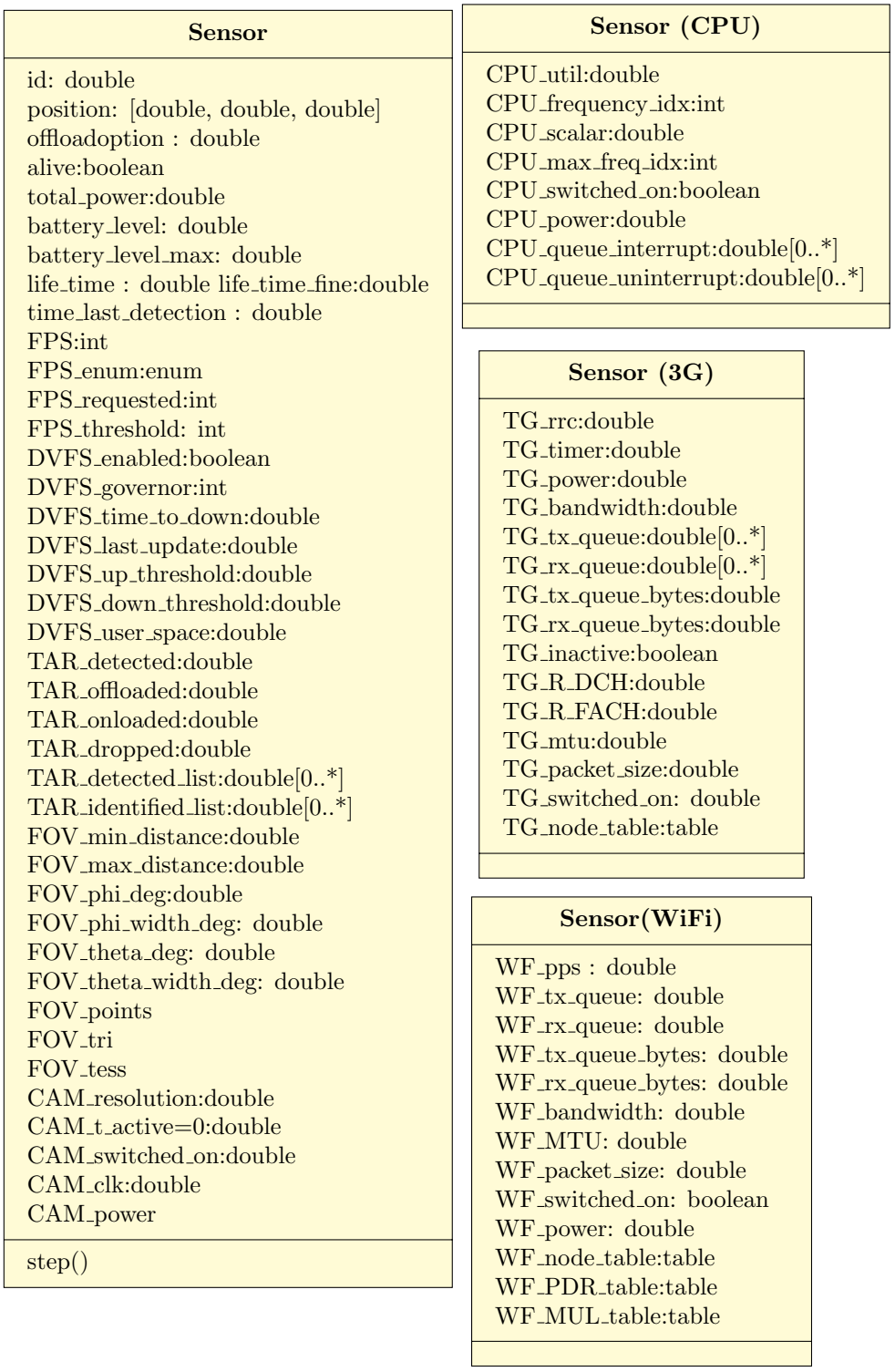


Figure B.1: Class diagram of Sensor showing important attributes. In the figure, it is split into three different classes for space

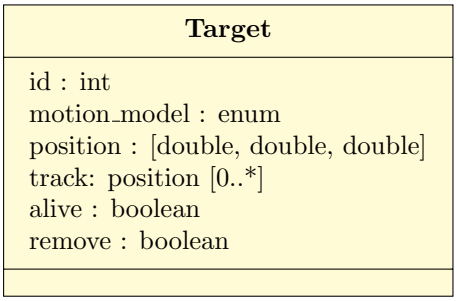


Figure B.2: Class diagram of Target Class

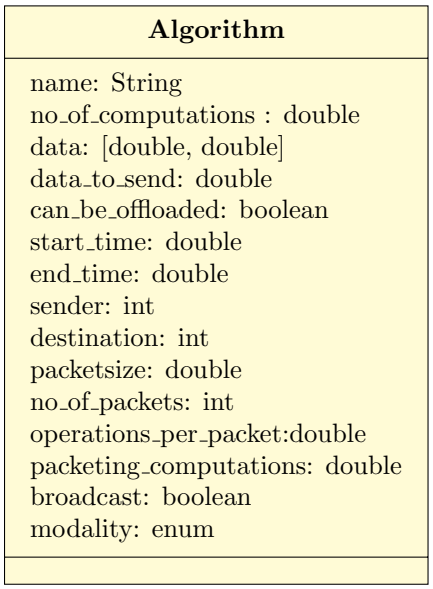


Figure B.3: Class diagram of Algorithm

B.3 Algorithm

Figure B.3 shows the properties of the Algorithm class that tells the simulator if the algorithm is offloadable or not, where it is originated from, and how much computation it requires, etc.

B.4 Platform

The platform class is an important entity of the simulator as it takes care of scheduling all the targets and sensors. In addition it also handles the communication between

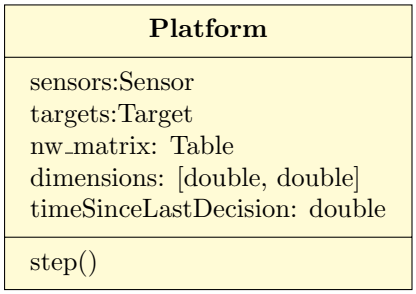


Figure B.4: Class diagram of Platform

sensors for various modalities. The class diagram of the platform is shown in Figure B.4. There are multiple instances of "Sensor" and "Target" class inside a platform.

Appendix C

Original Publications

C.1 Journal Paper

- S. Sthapit, J. Thompson, N. M. Robertson and J. R. Hopgood, "Computational Load Balancing on the Edge in Absence of Cloud and Fog," under review in IEEE Transactions of Mobile Computing

C.2 Conference Papers

- S. Sthapit, J. R. Hopgood and J. Thompson, "Distributed computational load balancing for real-time applications," 2017 25th European Signal Processing Conference (EUSIPCO), Kos, 2017, pp. 1385-1189. doi: 10.23919/EUSIPCO.2017.8081436

- S. Sthapit, J. R. Hopgood, N. M. Robertson and J. Thompson, "Offloading to neighbouring nodes in smart camera network," 2016 24th European Signal Processing Conference (EUSIPCO), Budapest, 2016, pp. 1823-1827. doi: 10.1109/EUSIPCO.2016.7760563

- S. Sthapit, J. Thompson, J. R. Hopgood and N. M. Robertson, "Distributed

Implementation for Person Re-Identification," 2015 Sensor Signal Processing for Defence (SSPD), Edinburgh, 2015, pp. 1-5. doi: 10.1109/SSPD.2015.7288501

Distributed Implementation for Person Re-identification

Saurav Sthapit*, John Thompson*, James R. Hopgood* and Neil M. Robertson†
*Institute of Digital Communications, School of Engineering, University of Edinburgh
Emails:{s.sthapit, john.thompson, james.hopgood}@ed.ac.uk
†Vision Lab, School of Engineering and Physical Sciences, Heriot-Watt University
Email: n.m.robertson@hw.ac.uk

***Abstract*—Person re-identification is to associate people across different camera views at different locations and time. Current computer vision algorithms on person re-identification mainly focus on performance, making it unsuitable for distributed systems. For a distributed system, computational complexity, network usage, energy consumption and memory requirement are as important as the performance. In this paper, we compare the merits of current algorithms. We consider three key algorithms, Keep It Simple and Straightforward MEtric (KISSME), Symmetry-Driven Accumulation of Local Features (SDALF) and Unsupervised Saliency Matching (USM). The advantage of SDALF, and USM is that they are unsupervised methods so training is not required but computationally many time expensive than KISSME. The Saliency based method is superior in performance but also has the largest feature size. As the features needs to be transmitted from one camera to other in distributed system, this mean higher energy consumption and longer time delay. Among these three, KISSME offers a balance between performance, complexity and feature lengths and hence more suitable for distributed systems.**

I. Introduction

Person re-identification refers to associating people across camera views at different locations and times [1]. It can have huge impact on surveillance and security because manual identification is not only tedious and costly but the results may also be received too late. The main challenges it faces is that the Field Of View (FOV) of the cameras can be non-overlapping, background and pose can change, as well as the occurrence of occlusion. A particular individual can look dissimilar in different views, while different individuals can look similar from different angles. Figure1 shows some sample pedestrian images from the VIPeR dataset [2] taken by two cameras illustrating these difficulties.

Fig. 1. Samples of pedestrian images from VIPeR dataset [2]

Person re-identification algorithms can broadly be classified into supervised and unsupervised algorithms. Supervised methods include algorithms like Mid-level features [3], Keep It Simple and Straightforward MEtric (KISSME) [4], Locally Aligned Featrue Transform (LAFT) [5], Information Theoretic Metric Learning (ITML) [6]. They mostly focus on metric learning, whereas unsupervised algorithms focus on feature design. Some of the unsupervised methods include Symmetry-Driven Accumulation of Local Features (SDALF) [7], Bio-inspired Covariance based features (BiCov) [8] and spatio-temporal [9]. For a more detailed review of recent approaches, refer to these papers [1], [10], [11], [12].

Current research in this area, however, focusses on implementing their algorithm on a single system [7], [4], [13], [3]. Implementing person re-identification on a distributed system has numerous benefits which will be illustrated with the example shown in Fig. 2. The system comprises of multiple smart cameras which may be static or moving. They are shown in the Fig. 2 by black and white camera icons respectively. The cameras are connected to each other and their field of view may be non-overlapping. The targets 1 and 2 are moving along the path shown by the arrows.

In a centralised system, all the sensor nodes would have been connected to a single computer with immediate access to data from all the sensor nodes. But on the downside, it has to process the data itself, which may be challenging particularly in real-time applications. In the distributed case, each sensor node has access to its own data only but offers more flexibility for signal processing. Running it on wireless embedded platform such as smartphone could be possible, which means the cameras could be deployed and scaled easily. In a military context, this means the camera may be embedded within a soldier's uniform to monitor targets without raising suspicion in conflict zones. We can think of light cameras in Fig.2 as these soldiers monitoring target 2. however, along with the algorithm's accuracy, there are several other factors to think about such as feature data length, computational complexity etc.

In this paper, we discuss the advantages and the disadvantages of current person re-identification algorithms when implemented on a distributed platform. The paper is structured as follows. Section II describes the basic workflow in person re-identification. Then we analyse various algorithms in section III. Section IV describes the experiments carried out and their results. Finally section V discusses the results and concludes the paper.

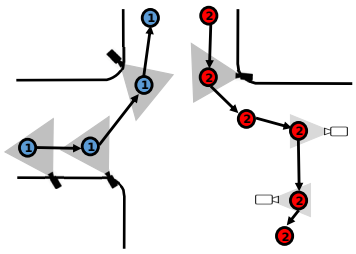


Fig. 2. Scenario of multi-camera person re-identification. Shaded cameras are fixed, white cameras are moving and grayed area represent Field of View (FOV)

978-1-4799-7444-3/15/$31.00 ©2015 IEEE

II. System Description

Person re-identification algorithms generally follow the basic workflow depicted in Fig. 3. Images are taken from each camera and preprocessed. The pre-processing step may include background subtraction and a person detection algorithm. To create a unique signature of each person, features are extracted. Popular features include combination of low level features such as colour histograms, Local Binary Patterns (LBP) [14], Scale Invariant Feature Transform (SIFT) [15] and Histogram of Gradient(HOG) [16]. Metric distance between signatures is calculated to verify if the images belong to the same individual or not. Alternatively, the test signature may be compared with the gallery set containing signatures of a seen individual to find the correct match. Some researchers have defined the person identification problem as a ranking problem [17].

In the distributed case, the signature has to be communicated from one camera to another as shown in the Fig. 3. Very often, these camera are connected with wireless networks such as Wi-Fi or cellular system. We know that the time taken and energy required to send the data across the network is directly proportional to the length of the data [18]. We conduct an experiment to quantize the energy and time required for such system in section IV-A.

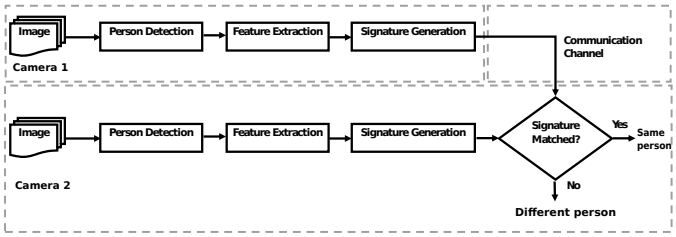


Fig. 3. Person Re-identification workflow.

Depending upon the number of images used, algorithms can be classified into single-shot and multi-shot algorithms. Single-shot algorithms take into account only one image per person (class) whereas multiple-shot algorithms uses multiple images. Multi-shot algorithms tries to keep the signature data size low and keep the matching considerably fast by throwing away redundant information.

A. Distributed Scenario

For implementing the re-identification system on a distributed system, let us assume each camera in Fig.3 has its own processing capability. So each sensor node can generate signature for the people in its FOV. For signature matching, one device has to send their signature to its neighbour so that it can be matched with its camera views. These are often battery powered devices, such as a smartphone, so longevity of the battery is desired. As it is desirable to keep the signature size as small as possible, we analyse the size of descriptors of the algorithms in consideration. Distributed systems are equipped with less powerful processors and have less memory resources, so the complexity of the algorithm is desired to be as low as possible. In order to measure complexity, we measure the time taken to run. For this paper, we have run our experiments on a desktop computer.

B. Datasets

Popular publicly available datasets for person re-identification are listed in Table.I. VIPeR is the most widely used and challenging dataset, one of the reason being limited samples per subject. We have used the VIPeR dataset in our experiments because many published algorithm comparisons are available.

III. Person Re-identification Algorithms

Among many algorithms, we have selected three key ones owing to their significance in person re-identification and availability of their source code. We go through them very briefly here.

A. KISS MEtric Learning

Keep It Simple and Straightforward Metric (KISSME) [4] focusses on learning the metric rather than complicated descriptor design. For the descriptor, images are divided into overlapping blocks and histograms are extracted in HSV and LAB colour-space. Local Binary Patterns (LBP) [14] are extracted to capture the texture information. For the VIPeR dataset, based on the code and data[1] provided by authors [4], each image has 22154 dimension features. Principal Component Analysis (PCA) is used by the authors to shorten the length of the descriptor to 34 experimentally chosen dimensions.

The Mahalanobis Metric learning is a widely used method in classification and in computer vision. It is defined as the squared distance between two points x_i and x_j as

$$d_M^2(x_i, x_j) = (x_i - x_j^T)\boldsymbol{M}(x_i - x_j) \tag{1}$$

where $\boldsymbol{M} \succeq 0$ is a positive semi-definite matrix.The main approach of Mahalanobis based algorithms is to define and learn the matrix $\boldsymbol{M}$ such that distance between images of same class is minimised and distance between images of different classes are maximised. KISSME [4], ITML, [6], LDML [21] and LAFT [5] are based on these methods. A detailed review of Mahalanobis based methods can be found in Roth et al's paper [22]. KISSME tries to address the metric learning approach from a statistical inference point of view. They test the hypothesis H_0 that the pair is dissimilar versus the alternative hypothesis H_1 that the pair is similar.

$$\delta(\mathbf{x}_{ij}) = \log\left(\frac{p(\mathbf{x}_{ij}|H_0)}{p(\mathbf{x}_{ij}|H_1)}\right) = \log\left(\frac{f(\mathbf{x}_{ij}|\theta_0)}{f(\mathbf{x}_{ij}|\theta_1)}\right) \tag{2}$$

where $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$ is the pairwise difference with zero mean. A high value of $\delta(\mathbf{x}_{ij})$ means the pair are dissimilar and vice-versa. Assuming a Gaussian structure of the difference space, Eq. 2 can be written as

$$\delta(\mathbf{x}_{ij}) = \log\left(\frac{\frac{1}{\sqrt{2\pi|\Sigma_{y_{ij}=0}|}}\exp(-1/2\,\mathbf{x}_{ij}^T\Sigma_{y_{ij}=0}^{-1}\mathbf{x}_{ij})}{\frac{1}{\sqrt{2\pi|\Sigma_{y_{ij}=1}|}}\exp(-1/2\,\mathbf{x}_{ij}^T\Sigma_{y_{ij}=1}^{-1}\mathbf{x}_{ij})}\right) \tag{3}$$

where,

$$\Sigma_{y_{ij}=0,1} = \sum_{y_{ij}=0,1}(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T \tag{4}$$

They arrive at the Mahalanobis distance metric in Eqn.1 that reflects the properties of the log-likelihood ratio test by re-projecting $\hat{\boldsymbol{M}} = \left(\Sigma_{y_{ij}=1}^{-1} - \Sigma_{y_{ij}=0}^{-1}\right)$ onto the cone of positive semi-definite matrices.

[1] accessible from https://lrs.icg.tugraz.at/research/kissme/

TABLE I. POPULAR PERSON RE-IDENTIFICATION DATASETS

Dataset	No. of Person	No. of Images	Features
VIPeR [2]	632	1264	pose, background, only 1 image per subject per camera
CAVIAR4REID [19]	72	1220	pose, background, varying resolution, multiple images per subject per camera
CUHK01 [20]	971	3884	pose, background, multiple images per subject per camera

B. Symmetry-Driven Accumulation of Local Features(SDALF)

SDALF [7] is suitable for single-shot and multi-shot images. The pedestrian image is divided into the head, torso and leg region and three types of features Weighted Color Histograms(WHSV), Maximally Stable Color Region(MSCR) and Recurrent High-Structured Patches (RHSP) are extracted. Each of these features are extracted from the torso and leg region and optionally from the head region. The histograms feature is built with 12 bins channel per region, totalling to $12 \times 3 \times 3 = 108$ dimensions[2]. The MSCR feature of a blob is represented by 9 dimensional feature but these blobs per image is variable. Similarly, the feature length of RHSP features is variable as well. Similarity between two images is calculated as weighted sum of euclidean distance between their features. As the algorithm is unsupervised, it doesn't require any training and is also scalable to videos.

C. Unsupervised Saliency

Saliency is defined as "distinct features that 1) are *discriminative* in making a person standing out from their companions, and 2) are *reliable* in finding the same person across different views" [23]. Zhao et al. have developed a few variants of supervised and unsupervised methods using saliency [13], [23], [3] but we will mostly focus on Unsupervised Salience Matching [13]. Each image is densely divided into overlapping patches. For each patch, 32 bin LAB colour histograms are computed in three scales for three channels. So the colour feature is of length $32 \times 3 \times 3 = 288$. Similarly for SIFT features, each patch is further divided into 4×4 cells to obtain $4 \times 4 \times 8 = 128$ dimensional feature per channel. So total feature length for each patch is $288 + 128 \times 3 = 672$ dimensions. For an image, these DenseFeats features is represented as $X^{A,u} = \{x^{A,u}_{m,n} | m = 1..., M, n = 1..., N\}$ where (A, u) denotes the u^{th} image in camera A, (m, n) denotes the patch centred at the m^{th} row and the n^{th} column of the image. Total size of feature for an image is $M \times N \times 672$.

Once, the features are extracted for each patch, the key steps of the algorithm is briefly listed in Table II. Fig.4 illustrates the adjacency constrained search set of the patch in yellow box which is used in computing the Nearest Neighbour set. One of the two approaches is based nearest neighbour distances. A score is assigned for each patch using Eq. 5.

$$\mathbf{score}_{knn}(x^{A,u}_{m,n}) = D_k(X_{NN}(x^{A,u}_{m,n})) \tag{5}$$

where D_k denotes the distance of the k-th nearest neighbour. Similarity between two images is calculated using Eq.6

$$\mathbf{Sim}(\mathbf{x}^{A,u}, \mathbf{x}^{B,v}) = \sum_{m,n} \frac{\mathbf{score}_{knn}(x^{A,u}_{m,n}).s(x^{A,u}_{m,n}, x^{B,v}_{i,j}).\mathbf{score}_{knn}(x^{B,v}_{i,j})}{\alpha_{sdc} + |\mathbf{score}_{knn}(x^{A,u}_{m,n}) - \mathbf{score}_{knn}(x^{B,v}_{i,j})|} \tag{6}$$

IV. SIMULATION RESULTS

In the ideal scenario, the algorithms would be implemented on a real distributed system such as Android smartphone and results could be measured. However, the algorithms are initially written in MATLAB to simulate a distributed system scenario and the simulations were carried out on MATLAB running on a desktop PC. In future, we can experiment with implementing the algorithms on embedded device to check their performance.

[2]reduced to 72 if head region is not used

Fig. 4. Illustration of adjacency constrained search. Green region represents the adjacency constrained search set of the patch in yellow box. The patch in red box is the target match [23]

TABLE II. ALGORITHM FOR UNSUPERVISED HUMAN SALIENCY LEARNING

Algorithm for learning Unsupervised Human saliency
Input: image $X^{A,u}$ and a reference image set $\mathbb{R} = \{X^{B,v}, v = 1, ...N_r\}$
Output: saliency probability map $P(l^{A,u}_{m,n} = 1 \| x^{A,u}_{m,n})$
for each patch $x^{A,u}_{m,n}$ **do**
compute Nearest Neighbour (NN) set $X_{NN}(x^{A,u}_{m,n})$
compute $score_{knn}(\mathbf{x}^{A,u}_{m,n})$ based on NN distances,
end for

Experiments were carried out on a desktop PC with an Intel Xeon processor (X5650) with 12 cores and 24 gigabytes of RAM running Scientific Linux 6.5 unless specified. Some of the algorithms have parallel implementation as well but we have turned it off for these experiments for two reasons. 1) To make the comparisons fair, 2) Parallel MATLAB instances run within their own Java Virtual Machine (JVM) environments accounting for increased memory allocations. This caused some algorithms to fill the RAM to fill quickly and slowing down the execution.

For the experiments, the VIPeR dataset was randomly split into two sets of 316 image pairs each. One set was used for training and other for testing. We do this following the testing conventions in these papers [7], [4], [13].

A. Cost of sending data in wireless network

In the distributed case, the signature of a person extracted in one camera has to be transmitted to another via a communication channel as shown in Fig. 3. The implication of transferring data to a neighbour node has a cost in terms of energy and time, particularly in the case of wireless transmission. We conducted simple experiment to analyse how much energy and time is required in order to data to other nodes. We developed a simple application(app) for the Android platform which sends files of various sizes to the server using WiFi or mobile data (see Fig.5). The application was built using Google's Android Development Kit (ADK) and Android Studio. The experiments were conducted in a LG G2 smartphone. Time is measured using the system clock. Initial time is noted when data sending commences. The final time is noted after an acknowledgement is received from the server and the time taken

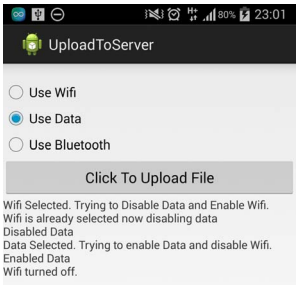


Fig. 5. Android application for calculating time and energy cost of transmitting data

is the difference of these two. Measuring energy consumed is however complicated than measuring time, because by default Android reports battery level in percentage only.It is too crude for our purpose and also as many processes are running simultaneously in background, it's hard to calculate the exact energy consumed for the communication. We used a third party application called Trepn profiler [24]. It is developed by Qualcomm for their Snapdragon processors and has access to hardware counters in the processor which are not available for public use. It isolates the energy used by an application, by collecting baseline energy consumption before starting the test application. Similar to the counter for measuring time, we flag the start and the end of the communication event to the Trepn application using Android Intent. Trepn then logs the energy consumption for each event.

As expected, the evaluations show in Fig.6 that the cost rises as the size of data goes up. WiFi has generally lower energy consumption than the phone networks. The difference becomes notable as the size of data goes up. Surprisingly, the speed of 4G was even faster than the WiFi albeit at higher energy cost. The test were done in Edinburgh with the WiFi provided by router connected to the Virgin Network and 4G by Everything Everywhere (EE) Network. But we didn't take into account many factors such as the load on the network, Signal strength etc.

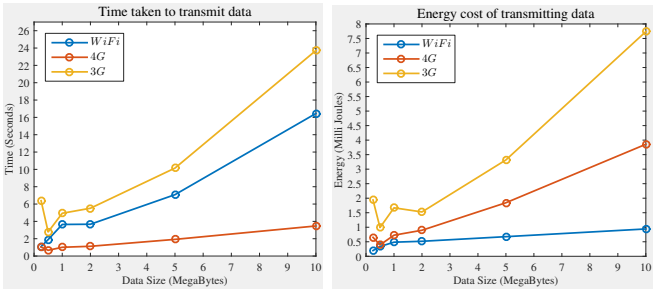


Fig. 6. Time and energy required to send data across the network

B. Runtime and Feature Length

1) KISSME: Among all the methods, KISSME was the fastest to train and learn the metric and it performed well too. The length of the feature before and after dimensionality reduction was determined from the source code and feature dataset provided. However, to calculate the time taken for feature extraction, we wrote the code as per their paper [4]. We divide the image into overlapping blocks of size 8×16 and stride of 8×8 to get 105 patches. We took histograms of 24 bins per channel and uniform LBP of 59 bins. So in total, the feature size is $105 \times 3 \times 2 \times 24 + 105 \times 59 = 21315$ dimensions. The histogram extraction of HSV and LAB and LBP features

TABLE III. SDALF EXECUTION TIME

Step	Time(sec)
Division into 3 parts	162.15
MSCR Extraction	138.21
WHSV Extraction	123.17
RHSP Extraction	4824.6
MSCR Matching	6095.3
WHSV Matching	214.74
RHSP Matching	423.00
Total	11981.17

TABLE IV. FEATURE LENGTH, RUNTIME AND RANK 1 RESULTS.

Algorithm	Feature Length(PCA)	Time(sec)	Rank 1
KISSME	22154(34)	260.05	18.03
SDALF	5359	11981.00	19.80
Unsupervised Saliency	201600	11737.90	27.22

took approximately 260 seconds, which is very high compared to its training time of around 0.05 seconds. But still, feature extraction per image would take about $260/1264 \approx 0.2$ seconds.After dimensionality reduction, the feature dimension is reduced to just 34 which is highly desirable.

2) SDALF: As discussed in section III, the feature length of SDALF is not fixed but dependent on the number of RHSP patches and MSCR regions found in the image. Table III shows the breakdown of average time spent per step for the VIPeR dataset. RHSP features took the longest to compute so we experimented with removing it. The result showed there was only marginal degradation of performance. It can be seen in Fig. 7. But as the test has been done only in one dataset, it may not be true for all.

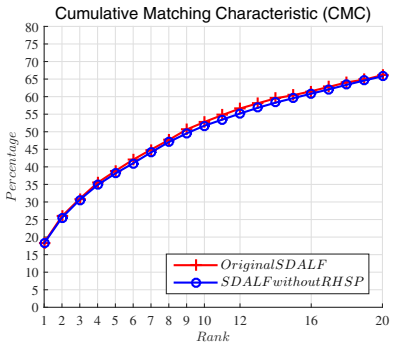


Fig. 7. Performance of SDALF with and without RHSP

3) Saliency: Saliency learning has the highest feature size per image. Each feature is of 201600 dimensions, if we suppose it is of MATLAB double precision, it's size is approximately 1.5 Megabytes which is not huge. However, each probe patch has it own adjacency search area for each image in the gallery set. If we assume 10 patches per row and constrained search area to be ± 2 rows, and there are 100 images in the gallery then. For each patch, we need to calculate the distance between itself and $10 \times 5 = 5000$ patches[3]. If there are 300 patches per image, it amounts to $5000 \times 300 = 1,500,000$ distances per image, which is more than 11 Megabytes in MATLAB double precision. In terms of running on embedded devices, memory is often a limited resource.

C. Cumulative Matching Characteristics (CMC) curves

Cumulative Matching Characteristics(CMC) [25] is widely used in person re-identification performance evaluation. It treats person re-identification as a ranking problem. Rank-1

[3]except for two top and two bottom rows

implies that the correct match has been found whereas Rank-k implies there were $k-1$ wrong classes ahead of the correct class. CMC(k) measures the probability that the correct match has a rank equal or higher than k [10]. TableIV shows Rank-1 score of various algorithms. It shows Saliency has better performance although it is computationally expensive and high data size. KISSME on the other hand looks the best to be implemented on distributed system as it is shown to be fast and computationally inexpensive as well.

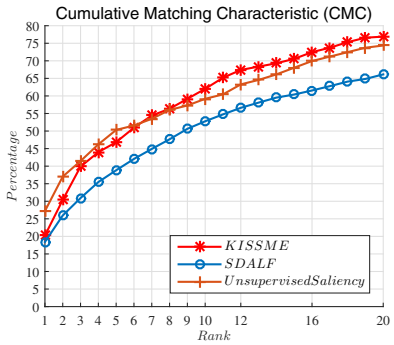


Fig. 8. Performance of the algorithms in VIPeR dataset

V. Conclusion

In this paper, we explored the possibilities of implementing person re-identification algorithms on distributed systems. We studied KISSME, SDALF and Unsupervised Saliency matching in terms of their runtime, size of descriptor, along with their person re-identification performance. We also looked at time and energy cost of communicating with neighbouring systems using various wireless technologies. Unsupervised Saliency has better Rank-1 result but it is computationally the most expensive and the memory requirement is also the highest. Even though we did not mention the energy cost for computing on the distributed platform, this would also consume high amount of energy. SDALF on the other hand has smallest signature before dimensionality reduction and potentially could be made even smaller by removing RHSP features. In theory at least, SDALF and Saliency features may be reduced using dimensionality reduction as well. But based on our experiments, without any modifications, KISSME is the best algorithm for a distributed system owing to its low complexity and shortest signature length. The only drawback is that it has to be trained and the large covariance matrices has to be computed and communicated to the neighbours.

This paper explored only the consequences of using distributed systems for person re-identification systems where communication between the sensor nodes is a requirement. But in some cases there might be a question between communicating or processing on its own. Even with communicating between nodes, there is a question of which node to communicate to when multiple nodes are available. In future, we are interested in answering these questions.

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. Gong, M. Cristani, S. Yan, and C. C. Loy, *Person Re-Identification, Advance in Computer Vision and Pattern Recognition*. 2014.

[2] D. Gray, S. Brennan, and H. Tao, "Evaluating appearance models for recognition, reacquisition, and tracking," *Perform. Eval. Track. Surveill. (PETS), 10th Int. Work.*, vol. 3, pp. 41–47, 2007.

[3] R. Zhao, W. Ouyang, and X. Wang, "Learning Mid-level Filters for Person Re-identification," *2014 IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 144–151, June 2014.

[4] M. Kostinger, M. Hirzer, P. Wohlhart, P. M. Roth, and H. Bischof, "Large scale metric learning from equivalence constraints," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, no. Ldml, pp. 2288–2295, 2012.

[5] Z. Li, S. Chang, F. Liang, T. S. Huang, L. Cao, and J. R. Smith, "Learning Locally-Adaptive Decision Functions for Person Verification," *2013 IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 3610–3617, June 2013.

[6] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon, "Information-theoretic metric learning," *Icml*, pp. 209–216, 2007.

[7] M. Farenzena, L. Bazzani, a. Perina, V. Murino, and M. Cristani, "Person re-identification by symmetry-driven accumulation of local features," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 2360–2367, 2010.

[8] H. Ma, C. Zeng, and C. X. Ling, "A Reliable People Counting System via Multiple Cameras," *ACM Trans. Intell. Syst. Technol.*, vol. 3, pp. 1–22, Feb. 2012.

[9] T. B. T. Sebastian, P. H. Tu, J. Rittscher, R. Hartley, and N. Gheissari, "Person Reidentification Using Spatiotemporal Appearance," in *2006 IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. - Vol. 2*, vol. 2, pp. 1528–1535, IEEE, 2006.

[10] X. Wang and R. Zhao, "Person Re-Identification: System Design and Evaluation Overview," in *Pers. Re-Identification* (S. Gong, M. Cristani, S. Yan, and C. C. Loy, eds.), London: Springer London, 2014.

[11] M. H. M. Saad, H. B. Zaman, M. A. Saghafi, and A. Hussain, "Review of person re-identification techniques," *IET Comput. Vis.*, pp. 1–20, Feb. 2014.

[12] A. Bedagkar-Gala and S. K. Shah, "A survey of approaches and trends in person re-identification," *Image Vis. Comput.*, vol. 32, pp. 270–286, 2014.

[13] R. Zhao, W. Ouyang, and X. Wang, "Unsupervised Salience Learning for Person Re-identification," *2013 IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 3586–3593, June 2013.

[14] T. Ojala, M. Pietikäinen, T. Maenpaa, T. Mäenpää, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *Pattern Anal. Mach. Intell. IEEE Trans.*, vol. 24, no. 7, pp. 971–987, 2002.

[15] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, pp. 91–110, 2004.

[16] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, pp. 886–893, 2005.

[17] B. Prosser, W.-S. Zheng, S. Gong, and T. Xiang, "Person Re-Identification by Support Vector Ranking," *Procedings Br. Mach. Vis. Conf. 2010*, vol. 1, pp. 21.1–21.11, 2010.

[18] 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.

[19] L. B. Dong Seon Cheng, Marco Cristani, Michele Stoppa and V. Murino, "Custom Pictorial Structures for Re-identification," *Proc. Br. Mach. Vis. Conf.*, pp. 68.1—-68.11, 2011.

[20] W. Li and X. Wang, "Locally aligned feature transforms across views," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 3594–3601, 2013.

[21] M. Guillaumin, J. Verbeek, and C. Schmid, "Is that you? Metric learning approaches for face identification," *2009 IEEE 12th Int. Conf. Comput. Vis.*, pp. 498–505, 2009.

[22] P. M. Roth, M. Hirzer, M. Köstinger, C. Beleznai, and H. Bischof, "Mahalanobis Distance Learning for Person Re-identification," in *Pers. Re-Identification*, pp. 247–267, Springer London, 2014.

[23] R. Zhao, W. Ouyang, and X. Wang, "Person Re-identification by Salience Matching," *2013 IEEE Int. Conf. Comput. Vis.*, pp. 2528–2535, 2013.

[24] Q. Technologies, "Trepn Profiler Starter Edition User Guide." 2014.

[25] J. R. Beveridge, D. S. Bolme, B. A. Draper, and M. L. Teixeira, "The CSU face identification evaluation system - Its purpose, features, and structure," *Mach. Vis. Appl.*, vol. 16, no. April 2003, pp. 128–138, 2005.

Offloading to Neighbouring Nodes in Smart Camera Network

Saurav Sthapit*, James R. Hopgood*, Neil M. Robertson† and John Thompson*
*Institute of Digital Communications, School of Engineering, University of Edinburgh
Emails:{s.sthapit, james.hopgood, john.thompson}@ed.ac.uk
†Vision Lab, School of Engineering and Physical Sciences, Heriot-Watt University
Email: n.m.robertson@hw.ac.uk

***Abstract*—Mobile Cloud Computing refers to offloading computationally intensive algorithms from a mobile device to a cloud in order to save resources (time and energy) in the mobile device. But when the connection to the cloud is non-existent or limited, as in battle-space scenarios, exploiting neighbouring devices could be an alternative. In this paper we have developed a framework to offload computationally intensive algorithms to neighbours in order to minimise the algorithm completion time. We propose resource allocation algorithms to maximize the performance of these systems in real-time computer vision applications (drop less targets). Results show significant performance improvement at the cost of using some extra energy resource. Finally we define a new performance metric which also incorporates the energy consumed and is used to compare the offloading algorithms.**

***Index Terms*—Offloading, Mobile Cloud Computing, Energy**

I. Introduction

Off-the-shelf smartphones are becoming ubiquitous and powerful, making them an interesting prospect to form a smart networked camera. However, they are not powerful enough for many applications, especially if the results are required in real-time. We have previously considered their pros and cons for distributed person re-identification [1]. If smartphones are deployed to carry out computationally intensive computer vision tasks, such as person tracking and re-identification between multiple cameras, they may not always be able to process everything within a user specified time. As such we can define performance of a system as the ratio of number of jobs processed to the number of jobs available.

Conventionally, computationally intensive algorithms have been offloaded to the "cloud" and it has been shown in some cases to save time and energy [2] [3]. In this paper however, we present a novel framework to offload these tasks to neighbouring mobile nodes which can significantly increase the performance without substantially depleting battery resource compared to the non-offloading case. We also present a single metric called Efficiency Score (ES) which also incorporates the energy consumption along with the performance.

A. Computing platform types

If there is no network connectivity, the only option is to do on-board processing. However, if there is some connectivity, we have the option to offload. We use the term "*onloader*" for the system which the "*offloader*" offloads its workload to. Some on-board processing can reduce the amount of data to be communicated while freeing up the onloader's resources. For example, a background subtraction algorithm can limit the sensor from sending images with little or no activity. This saves communication cost for the sensor and the onloader has fewer jobs to perform. However, when the algorithms are fairly complex, the cloud is the preferred option. High round-trip latency can be compensated by its shorter runtime owing to high performance computing resources in the cloud. More details on mobile cloud offloading can be found in [4].

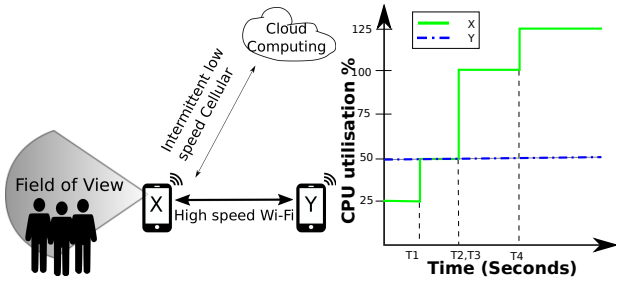


Fig. 1. Pedestrian identification scenario: device X inundated with targets while device Y is idle (left), shows computational load on X and Y. T1–T4 indicates arrival of targets (right).

Neighbouring nodes are important alternative to the cloud for two reasons, the first reason is communicating with neighbours can help in the co-ordination and control of the node network. Neighbours can provide a cue of an incoming target, or give complementary and valuable information about the targets. More details about co-ordination and control can be found in [5]. The second reason is that when the cloud is unavailable, they can help in sharing the computational load. For example, in an underground transportation network, battle-space scenario, or a search and rescue mission after natural disaster, the internet may be unreachable. Even if there is a connection, the Network Bandwidth (BW) may be too low or intermittent. Neighbouring devices lack the computational power and energy of the cloud but may be readily available with high BW connections. However, we need to consider their available energy resources before offloading a job as when in the field, charging may not be readily available. In this paper, we discuss its feasibility in terms of time and energy. Magurawalage et al. have considered offloading to intermediate cloud like entity called cloudlets [6] but to the best of our knowledge, battery powered neighbours have not been used by anyone as an offloading candidate.

B. Problem formulation

Let us assume there are two similar smartphones (X and Y) deployed in the field to identify people arriving in their Field Of View (FOV). Each can only process N targets at a time. When there are $P > N$ potential targets to process, as shown in the Fig. 1 where $N = 3$, X has no other option than to offload or drop some of the targets. If the cloud is available and the bandwidth is sufficiently high, the cloud can be the

978-0-9928-6265-7/16/$31.00 ©2016 IEEE

onloader. Then, X can offload $(P-N)$ targets to the cloud and as Kumar et. al explains, it can save time and energy [2]. If the cloud is not available, conventional system would simply drop the targets. However, device Y has no target in its FOV at this moment. We show that neighbour devices like Y can be good alternative and may be used for offloading.

We now describe the simulator we developed to study smartphones with cellular and Wi-Fi communication capabilities. Then in section III, we describe how we use the simulator for a person re-identification system. Then we describe experiments in section IV and show results. Finally in section V, we present conclusions to the paper.

II. Modelling network of smart cameras

The simulator allows us to use a simplified model of the algorithm flow for the target platform and update components easily as required. Wu et al. used queuing model theory to simulate workload on distributed nodes [3]. However, their assumption that when there is no workload the nodes do not consume any energy is not valid in real life. The major elements of our simulator relate to the algorithmic tasks, the sensor architecture, communication links and the targets. We go through each one in detail below.

A. Algorithmic tasks

The simulator's model for the algorithmic task is characterised by its number of Operations (OP), 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 OP per image and outputs the number of persons in the image. Assuming one OP per clock cycle, we can estimate the execution time on the device using the clock frequency.

$$T_{\text{exec}} \propto \frac{C}{\text{Clock Frequency}} \tag{1}$$

We are aware that in different processors some OP take more than one cycle and multiple OP can be possible in one cycle, however this approximation (Eqn. (1)) gives an estimate of time required without detailed execution information. If desired, algorithms could be executed on the Device Under Test (DUT) to measure the execution time more precisely.

The number of OP required for an algorithm can change. For instance, in the Mixture of Gradients (MOG) algorithm for background subtraction, it depends on how quickly a matching Gaussian distribution is detected for the particular pixel [7]. To make calculations easier for the simulation, we take the worst case scenario where the matching Gaussian is not found.

B. Component Based Sensors

In order to realistically emulate its behaviour, a sensor is divided into its components such as the Central Processing Unit (CPU) and cellular radio. We do not consider the energy consumption by the display as it can be turned off by the application. We use the utilisation based model by Jung et al. to calculate the energy consumption [8] and our parameters are based on a Google Nexus I phone which was one of their DUTs. However if desired, the simulator can be calibrated for a different DUT in a straightforward manner.

TABLE I. CPU Parameters

Frequency	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

1) Image Sensor: The image sensor consumes significant energy in a mobile device when used continuously. According to Likamwa et al., the energy consumption per frame of the image sensor can be modelled as follows [9].

$$E_{\text{camera}} = P_{\text{idle}} \times (T_{\text{frame}} - T_{\text{active}}) + P_{\text{active}} \times T_{\text{active}} \tag{2}$$

where $T_{\text{active}} = \frac{\text{Number of Pixels}}{\text{Camera Clock Frequency}}$. Based on Eqn. 2 , we can either reduce the image resolution, thereby reducing T_{active} or reduce the acquisition rate to save the energy consumption.

2) Application Processor (AP): The CPU power is made up of two parts, idle power and the running power, as follows:

$$p^{\text{cpu}} = \beta^{\text{cpu}}_{\text{freq}} \times u + \beta^{\text{cpu}}_{\text{idle}}, \tag{3}$$

where u is the utilisation and $\beta^{\text{cpu}}_{\text{freq}}$ and $\beta^{\text{cpu}}_{\text{idle}}$ are the CPU parameters listed in Table I. We calculate the utilisation as the ratio of the CPU time used vs the time available per frame. However, the CPU is also used by the Operating System (OS) and other running applications. Dargie used normal and exponential distributions to simulate workload in [10]. We also used a random variable (r) sampled from a Gaussian distribution to simulate these other activities. By adjusting the mean of r we can a simulate busy sensor and idle sensor. The total utilisation is calculated as shown below.

$$u = \frac{\sum_{i=1}^{N} T_{\text{exec}\,i}}{T_{\text{Frame}}} + r \tag{4}$$

where N is the number of algorithms to be processed, T_{exec_i} is the execution time for i^{th} algorithm (see Table II 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; we only run the CPU to 100% load and run the remainder of the algorithm in the next frame and so on.

3) Cellular (3G): Cellular radio is modelled as a three state system: IDLE, Forward Access Channel (FACH) and Dedicated Channel (DCH). The IDLE mode is the non communicating mode and has the lowest power consumption. In this mode, the User Equipment (UE) is turned on but has not established Radio Resource Control (RRC) connection with the Radio Network Controller (RNC). In DCH state the UE has a dedicated transport channel for data transmission in both directions, but this is 50 to 100% more expensive than FACH, where FACH is the intermediate state with reduced power consumption and low data rate. There is no dedicated channel allocated in this mode and it can only transmit user data through shared low speed channel that is typically less than 15kbps [11]. As we can see from Eqn. (5), power is only dependent on state but not on utilisation. Fig. 2 shows the state diagram with the inactivity timers which along with data buffer size controls the state promotions and demotions.

$$p^{3g} = \begin{cases} \beta_{\text{IDLE}} & \text{if RRC state is IDLE} \\ \beta_{\text{FACH}} & \text{if RRC state is FACH} \\ \beta_{\text{DCH}} & \text{if RRC state is DCH} \end{cases} \tag{5}$$

where RRC is the current state of UE and β_{IDLE}, β_{FACH} and β_{DCH} are based on [8].

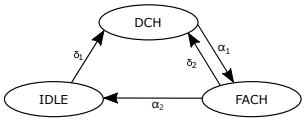


Fig. 2. Cellular radio states, α_1 and α_2 are inactivity timers whereas δ_1 and δ_2 are delay to get to DCH

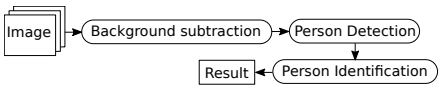


Fig. 3. Person Re-identification work-flow.

4) 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.

$$p^{\text{wifi}} = \begin{cases} \beta_{\text{LT}} \times p + \beta_{\text{LT base}} & \text{if } p \leq \text{Threshold} \\ \beta_{\text{HT}} \times p + \beta_{\text{HT base}} & \text{if } p > \text{Threshold} \end{cases} \quad (6)$$

where p is the packet rate, $\beta_{\text{LT}}, \beta_{\text{HT}}, \beta_{\text{LT base}}$ and $\beta_{\text{HT base}}$ are the parameters of the DUT based on [8]. 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. Although Wi-Fi consumes energy in scanning mode, we ignore it as we assume a connection between the sensors as the basis of this research.

III. System Design

In this section, we show how offloading may be used to increase the performance of a system. We consider a pedestrian re-identification system outlined in Fig. 3. It starts with image acquisition from the image sensor. A background subtraction and a person detection algorithm is applied on the image to detect the number of people in the view. When there a pedestrian is detected, we apply a person re-identification algorithm to each detection such as [12]. Our goal is to identify as many detections as possible.

A. Application Partitioning

The algorithmic complexity of the person re-identification algorithm outweighs that of other algorithms in the chain (see Table II). So, the overall complexity of the system can be estimated as $\mathcal{O}(N)$ where N is the number of people detected. To be realistic, we limit the number of people in an image (800×600) in the simulator to be fewer than 10.

B. Energy Saving Methods

We replicate following energy saving techniques to make the simulation realistic as much as possible.

1) Dynamic Frame per Second: We can save energy by decreasing the number of FPS of the system (see Eqn. (4)). However, very low FPS may mean some of the detections may be missed. We implemented an algorithm to vary the FPS of each individual sensor between 1 and 16 in the following way.

$$\text{FPS (new)} = \begin{cases} \text{FPS (old)} \times 2 & \text{if } t < \tau \\ \text{FPS (old)} \div 2 & \text{if } t > \tau \end{cases} \quad (7)$$

where t is the time between target activities and τ is 5 seconds.

TABLE II. Execution times for CPU running @ 998.4 MHz

Algorithm	Execution Time
Background Subtraction	0.1
Person Detector	0.2
Person Re-identification	5.1
Total Time	5.3

2) Dynamic Voltage and Frequency Scaling (DVFS): A simple algorithm controls the clock frequency of the sensor. When the CPU utilisation is below 0.4, the clock frequency is lowered according to Table I and it is scaled to maximum frequency as soon as the utilisation is above 0.9.

C. Offloading

We classify only the re-identification algorithm as *offloadable* as for others, the communication costs and the time delay outweighs the benefits of offloading. Offloading an algorithm entails sending input data, waiting for the onloader to execute, and receiving output data. Before transmitting however, the data has to be formatted in packets and some overhead will be added to the processor. These operations can be a few hundred per packet which needs to be added to the CPU workload.

1) Time Cost: The communication times are proportional to the data size to be communicated and inversely proportional to the network BW. We assumed the BW to be static. Waiting times can be estimated using Eqn. (1) for the onloader's clock frequency. But, it does not take into account the CPU load. If our onloader's AP is already busy, our estimation can be very far from reality. So we re-write the time calculation using onloader's average CPU utilisation.

$$T_{\text{wait}} = \frac{T_{\text{exec}}}{1 - \mathbb{E}[u]} \quad (8)$$

where $\mathbb{E}[u]$ is the average of u from Eqn. (4). The total time cost which is also known as makespan, is shown below [13].

$$T_{\text{total}} = T_{\text{packet}} + T_{\text{send}} + T_{\text{wait}} + T_{\text{receive}} \quad (9)$$

where T_{packet} is the time to format the data in a packet.

2) Energy Cost: There are two energy costs involved. The first is for the *offloader* (E_{off}) and includes data packeting and the radio communication cost.

$$E_{\text{off}} = (T_{\text{send}} + T_{\text{receive}}) \times P_{\text{radio}} + P_{\text{cpu}} \times T_{\text{packet}} \quad (10)$$

where radio $\in \{3G, WiFi\}$. Second is for the onloader which includes radio cost, execution cost and the packeting costs. So far, in literature, cost for the *onloader* is ignored as energy is not of major concern for the cloud. But while offloading to the neighbours, we need to consider it.

$$E_{\text{on}} = (T_{\text{receive}} + T_{\text{send}}) \times P_{\text{radio}} + P_{\text{cpu}} \times (T_{\text{execute}} + T_{\text{packet}}) \quad (11)$$

D. Multi-Objective Optimisation

The time and energy costs from Eqn. (9, 10 and 11) can be inferred as variables of a multi-objective optimisation problem.

$$\text{Cost} = w_{\text{time}} \times T_{total} + w_{\text{off}} \times E_{\text{off}} + w_{\text{on}} \times E_{\text{on}} \quad (12)$$

where w_{time}, w_{off} and w_{on} are the weights for each objective. The cost function involves adding time and energy variables (i.e. different units), which requires careful selection of the

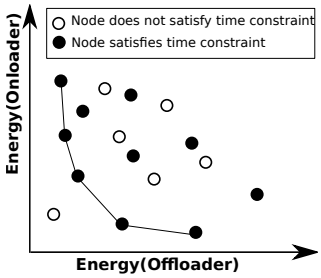


Fig. 4. Multi-Objective optimisation problem reduced to two objectives.

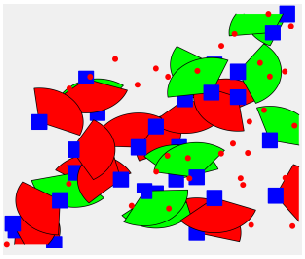


Fig. 5. Snapshot of simulation showing targets as dots and sensors. (Green FOV signifies target in FOV whereas red means no target in FOV).

weights. We avoid this situation by limiting one of the objectives to a threshold (ε) and optimising rest of the objectives [14]. Regarding the real-time nature of our problem, we limit the time and optimise the energy variables. We set $\varepsilon = 25$ seconds and leave out the nodes that do not satisfy this constraint (denoted by the light dots in Fig. 4). It is still a multi-objective problem but only with two variables of the same unit (Joules). We now study three methods to optimize offloading performance.

1) Minimize Energy Cost (MEC): In this method, we choose the node that satisfies the time constraint described above and incurs the minimum offloader and the onloader energy cost. The solution is pareto-optimal and denoted by nodes on the line in Fig. 4 [14].

$$\text{Cost}_{\text{MEC}} = w_{\text{off}} \times E_{\text{off}} + w_{\text{on}} \times E_{\text{on}} \tag{13}$$

2) Minimize Battery Impact (MBI): In a battery-powered device, using the least energy cost per job alone may not increase device lifetime. For example, say an algorithm requires 10 and 8 Joules on devices X and Y respectively. But X and Y have 500 and 50 Joules left in their battery respectively. Considering energy cost alone, Y is the best choice but when we consider the amount of energy left in the device clearly X is a better choice. We re-write Eqn. (13) as follows.

$$\text{Cost}_{\text{MBI}} = w_{\text{off}} \times \frac{E_{\text{off}}}{E_{\text{rem. off}}} + w_{\text{on}} \times \frac{E_{\text{on}}}{E_{\text{rem. on}}} \tag{14}$$

where $E_{\text{rem. off}}$ and $E_{\text{rem. on}}$ are the energy left in the offloader and the onloader nodes.

3) Offload Only if Busy (OOB): The previous methods try to find the global solution, but offloading has overhead costs. So, this method tries to offload only if on-board processing is estimated to be infeasible. To do so, we add all the operations in the execution queue and use Eqn. (1) to estimate the minimum remaining processing time. If this time is greater than the threshold (ε), offload the algorithm minimising the time and energy objectives defined in Eqn. (12).

IV. Simulation and Results

We simulated a number of sensors connected to each other by Wi-Fi and to the server (when available) by cellular link. For simplicity, we assume that resource information about all the nodes (remaining energy, current CPU load etc.) is available and all the sensors have same computational capability but the server is 10 times more powerful. Also there is no energy limitation for the cloud so the weight w_{on} for the cloud is set to zero. Wi-Fi is set to 10 Mbps (high BW) whereas cellular is slower and set to 1 Mbps. At the start of the simulation, the battery level is uniformly distributed between $0-10$ Watt-hour. The mean of r in Eqn. (3) is uniformly distributed from $0-1$ (full load) and the standard deviation is fixed to 0.1. These parameters do not change during the simulation. Full simulation data and parameters are available here [1]. Fig. 5 shows a snapshot of the simulation, where the blue squares and red dots represent the sensors and targets respectively . The targets are generated using a Poisson distribution and follow the random waypoint model [15]. In this model,targets move from one point to another with random speed and can also pause for random amount of time. When they enter into the FOV of the sensors (shown as green areas in Fig. 5), they are tagged as detected targets. The detected targets are then identified. Once the target has been detected and identified, it does not have to be re-identified again in the same camera. This is done to simulate tracking the target in each camera and using the best representation for identification purposes.

TABLE III. Simulation Results (Averaged over 100 runs)

Algorithm	Targets Detected	Targets Offloaded	Targets Dropped	SI	Energy Used (Joules)	ES (per 100J)
Cloud not available						
NO	**13.29**	0	2.91	10.38	**499.61**	2.08
MEC	13.28	5.19	0.61	12.67	546.05	2.32
MBI	**13.29**	12.5	4.42	8.87	572.31	1.55
OOB	13.28	5.18	**0.24**	13.04	546.88	2.38
Cloud available						
NO	**13.28**	0	2.90	10.38	**506.31**	2.05
MEC	13.25	13.13	**0.001**	13.25	651.28	2.03
MBI	13.25	13.23	0.002	13.25	652.23	2.03
OOB	13.26	4.95	0.18	13.08	601.58	2.17

We tested the algorithms with various parameters using 100 Monte-Carlo runs each representing a 10 minute period. The results are listed in Table III. When the cloud is not available, Successful Identifications (SI) (which is targets detected minus targets dropped) improved from 10.4 in the Non Offloading (NO) case to 12.7 for the MEC case and 13 for OOB case but degraded to 8.87 for MBI. MEC and OOB boosted the performance by more than 20% while only incurring around extra 10% energy consumption. MBI did not perform well because of the communication overhead. However, if the runs were longer, may be we would see some improvement in the device lifetime. All the algorithms did better than the NO case when the cloud was available, dropping almost no targets but the energy consumption was significantly higher. This shows that offloading to cloud blindly may increase the performance short term but may shorten device lifetime.

[1] http://sauravsthapit.com.np/EUSIPCO2016/

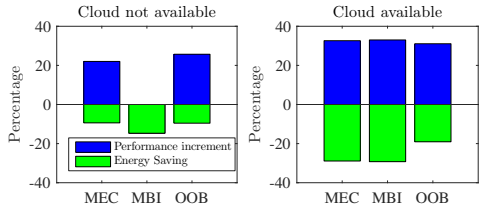


Fig. 6. Performance increment and energy saving in comparison to the NO.

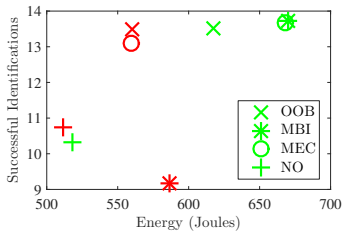


Fig. 7. Efficiency score. (red = no cloud available, green = cloud available)

A. Metrics

In order to access the performance of the offloading decision algorithms, we used two approaches. First we compared the algorithm's performance to the NO case. We calculate the performance of proposed algorithms in terms of performance improvement and energy savings. Fig.6 shows that both MEC algorithm and OOB algorithm has superior performance in all the scenarios. The MBI algorithm however did not fare very well when cloud was not available. Second, we define and calculate Efficiency Score (ES) of the algorithms as follows:

$$\text{ES} = \sum_{i=1}^{\text{Nodes}} \frac{\text{SI}_i}{\text{Energy Used}_i} \tag{15}$$

This can be interpreted as SI per Joule and means how productively the energy resources have been used. However, this should not be confused with the accuracy and energy consumption of the person re-identification algorithm. Table III shows OOB has the best ES score with 2.38 and MBI has the worst with 1.55. This is consistent with our intuition that the algorithm performing the best and with relatively lower energy consumption is better. In the *cloud available* case, ES score for NO is greater than for MEC and MBI which suggests that even though performance has improved, the energy is not used efficiently. The metric can be visualised in Fig. 7. It is desired to develop an algorithm with ES at the top left corner of the graph, which indicates low energy usage and high SI, whereas being in bottom right corner indicates high energy usage without fruitful performance.

V. Conclusion

This paper presented a simulation model for offloading computationally intensive algorithms to neighbouring devices when the cloud is not available. The results show that among the three, OOB consistently achieved the best trade-off between power and performance. It improved the performance by approx. 25% while costing about $10-20\%$ more energy. The ES metrics suggests that energy is used more productively. Contrary to the general belief, the results show that given the constraints in bandwidth offloading to the cloud may not be the best option in terms of performance and energy cost. In future work, we plan to evaluate more 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. Simulation information available at [2]

References

[1] S. Sthapit, J. Thompson, J. Hopgood, and N. Robertson, "Distributed implementation for person re-identification," in *Sensor Signal Processing for Defence, 2015*, pp. 1–5, Sept 2015.

[2] 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.

[3] H. Wu, W. Knottenbelt, and K. Wolter, "Analysis of the Energy-Response Time Tradeoff for Delayed Mobile Cloud Offloading," in *Teletraffic Congr. (ITC 27), 2015 27th Int.*, pp. 134–142, 2015.

[4] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Futur. Gener. Comput. Syst.*, vol. 29, pp. 84–106, jan 2013.

[5] P. Natarajan, P. K. Atrey, and M. Kankanhalli, "Multi-Camera Coordination and Control in Surveillance Systems : A Survey," *ACM Trans. Multimed. Comput. Commun. Appl.*, vol. 11, no. 4, 2015.

[6] C. M. Sarathchandra Magurawalage, K. Yang, L. Hu, and J. Zhang, "Energy-efficient and network-aware offloading algorithm for mobile cloud computing," *Comput. Networks*, vol. 74, pp. 22–33, 2014.

[7] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction," *Proc. 17th Int. Conf. Pattern Recognition, 2004. ICPR 2004.*, no. 2, pp. 28–31 Vol.2, 2004.

[8] 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.

[9] R. Likamwa, B. Priyantha, M. Philipose, L. Zhong, and P. Bahl, "Energy characterization and optimization of image sensing toward continuous mobile vision," in *Proceeding 11th Annu. Int. Conf. Mob. Syst. Appl. Serv.*, (New York, NY, USA), pp. 69—-82, ACM, 2013.

[10] W. Dargie, "A Stochastic Model for Estimating the Power Consumption of a Processor," *IEEE Trans. Comput.*, vol. 64, no. 5, pp. 1311–1322, 2015.

[11] C. W. Johnson, *Radio Access Networks for UMTS: Principles and Practice*. John Wiley and Sons, 2011.

[12] M. Farenzena, L. Bazzani, A. Perina, V. Murino, and M. Cristani, "Person re-identification by symmetry-driven accumulation of local features," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 2360–2367, 2010.

[13] Y. Jiang and S. Member, "A Survey of Task Allocation and Load Balancing in Distributed Systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 585–599, 2016.

[14] M. Caramia and P. DellOlmo, "Multi-objective Optimization," in *Multi-objective Manag. Freight Logist. Increasing Capacit. Serv. Lev. Saf. with Optim. Algorithms*, pp. 11–37, Springer London, 2008.

[15] F. Bai and A. Helmy, "A Survey of Mobility Models in Wireless Adhoc Networks," *Wirel. Ad Hoc Sens. Networks*, pp. 1–30, 2004.

[2] http://sauravsthapit.com.np/EUSIPCO2016/

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.

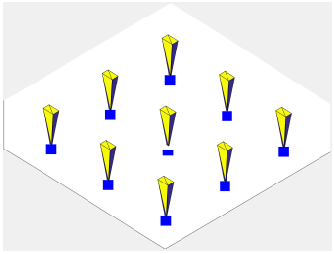


Fig. 1: Nine Camera sensors and their FOV. Blue square represents camera in a drone or a smart phone

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

ISBN 978-0-9928626-7-1

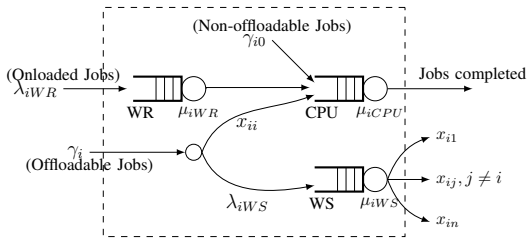


Fig. 2: A sensor node modelled as network of queues. CPU, WR, WS represent CPU, WiFi Receiver and WiFi Sender queues respectively

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.

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

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

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

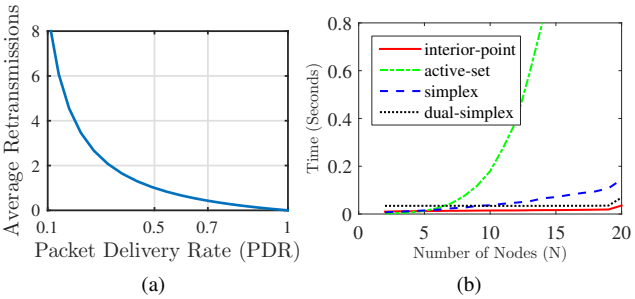
B_i, B_j are the remaining battery in node i, j

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.



(a) (b)

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

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

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

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

ISBN 978-0-9928626-7-1 © EURASIP 2017

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.

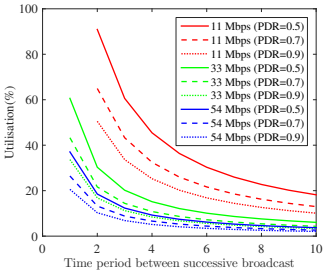


Fig. 4: Queue utilisation of the *server* in proactive under various network conditions and NSI update frequency. NSI size set to 1 Mb.

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 γ_i + γ_i0 ≤ μ_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 ≥ 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

ISBN 978-0-9928626-7-1

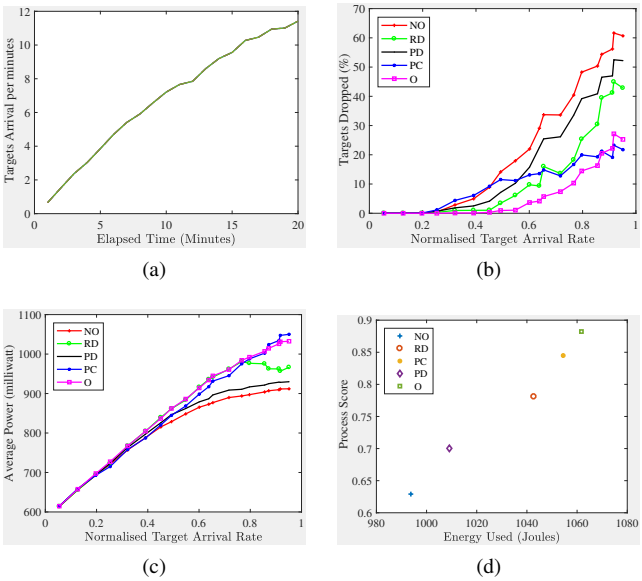


Fig. 5: (a) Target arrival rate per nodes over simulation time. (b) Targets dropped over Arrival Rate. (c) Power Consumed over Arrival Rate (d) Efficiency Score of proposed algorithms

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 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.

TABLE I: Simulation Results (Averaged over 100 runs)

Algorithm	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

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.

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. Simulations show slightly higher energy usage in comparision to the baseline non-offloading case, however, the *efficiency score* metric shows workdone per joule is at least as good as the baseline. 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 interesting 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. Reporting raw data back to a base station can be prohibitive in terms of both time and energy. Even worse, in a covert defence operation, 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, in this application, 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 scenarios or battlefields far from the base station.

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

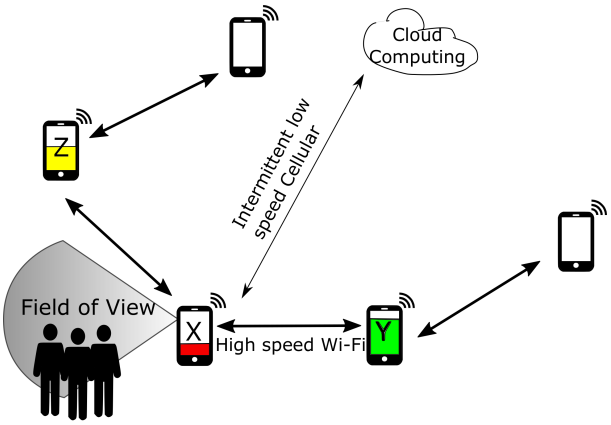


Fig. 1: Pedestrian identification scenario: device X inundated with targets while device Y is idle

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 may be valid even 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 the recharge time commences. In case of uneven load, by helping neighbours, the network life-time (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)(see section 3) that needs to be shared and its frequency, in order to make the scheduling decision. Queuing theory 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.

Given the model, the scheduling decision is posed as a minimum cost problem. Then a *purely distributed* solution is proposed 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, four novel algorithms are presented and their performance are compared with the non-offloading case. These 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:

- Propose novel algorithms for on-line workload balancing for real-time applications in distributed systems.
- Propose an 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 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 network of queues and the problem is formulated along with the Node State Information. In Section 4, the algorithms are proposed. Then 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 Mobile Cloud Computing

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 resources saving by offloading to the *cloud* [8], [9]. For a comprehensive list of MCC algorithms, interested readers should refer to the recent surveys [10], [11]. 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.

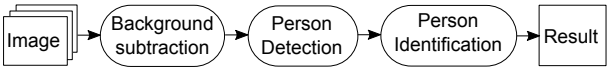


Fig. 2: Typical pedestrian identification flowchart

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) related algorithms and hardware related jobs cannot be offloaded.

Generally, MCC implementations use static and dynamic application partitioning of algorithms based on profiling [12]. For the current 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* algorithm as its time complexity far outweighs other in the chain [3].

2.2 Communication Channel

The availability and quality of a communication channel has 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. Wu et. al [13] 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 [14] 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 [15], [16]. 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. Also the cloud is mains powered, so not limited by energy consumption. 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. [17] considered offloading to *coprocessors* and Magurawalage et al. [18] considered offloading to cloudlets along with the *cloud*. Recently, Truong-Huu et al. [19] also considered smartphones as offloading candidates. Their main objective 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, do not mention 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 pros and cons of offloading to cloud vs offloading to neighbouring nodes are summarised in Table 1. The highlighted items show where each one may be superior to the other in a typical case. In case of higher bandwidth between neighbouring nodes is based on 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 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.

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 Queue network is presented in the next section. It would be useful in modelling of the sensor nodes.

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**

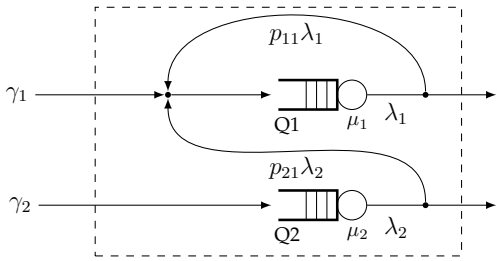


Fig. 3: A network of two Queues. Total incoming target rate at Q1 (λ_1) is the sum of external target rate (γ_1) and targets rates emanating from the queues heading to Q1. Under stable condition, outgoing rate is equal to the incoming rate.

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 [20]. 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 [20]. For example, Fig. 3 shows an open network with two $M/M/1$ queues Q1 and Q2 with external target rates γ_1 and γ_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,

γ_i is the rate of arrival of external targets at queue i,

λ_j is the arrival rate at queue j,

p_{ji} is the probability a job moves from queue j to i

Vilaplana et al. [21] 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. $M/M/1$ queues are used to model the behaviour of each of these components. Specifically, the $M/M/1$ has First Come First Service (FCFS) scheduling discipline, an

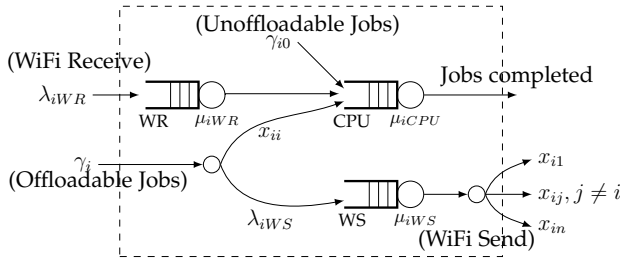


Fig. 4: A sensor node modelled as network of queues. CPU, WR, WS represent CPU, WiFi Receiver and WiFi Sender queues respectively

TABLE 2: List of Notation

Notation	Definition
N	Set of sensor nodes {1,...,n}
A	Set of directed arcs between nodes.
γ_i	Incoming external Offloadable jobs rate of i^{th} node
γ_{i0}	Incoming external Unoffloadable jobs rate of i^{th} node
λ_{iCPU}	Total incoming job rate for i^{th} CPU
μ_{iCPU}	Job service rate of CPU of i^{th} node
λ_{iWS}	Total incoming job rate of WiFi send queue for i^{th} node
μ_{iWS}	WiFi transmission rate of i^{th} node
λ_{iWR}	Total incoming job rate of WiFi receive queue for i^{th} node
μ_{iWR}	WiFi receive rate of i^{th} node
μ_{iWF}	Common WiFi 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

arrival process that is Poisson distributed, and a service time that is exponentially distributed [20]. Similarly, for the communication part, WiFi is modelled using two $M/M/1$ queues (sender and receiver side). Without any loss of generalisation, common WiFi send and receive rate (i.e $\mu_{iWS} = \mu_{iWR} = \mu_{iWF}$) is assumed. The resulting model of the node is depicted in Fig. 4. Each node i may 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 and μ_{iWF} is the WiFi transmission rate. 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 *non-offloadable jobs* may be spatially and temporarily distributed as well like the *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.

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

subject to

$$\sum_{j=1}^{n} x_{ij} = \gamma_i, \qquad \forall i \in N \tag{2b}$$

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

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

The decision variable $x_{ij} \in R^{(n \times m)}$ 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 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 (2b) makes sure that all the jobs are assigned to a processing node whereas the inequality constraint in (2c) 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. (2) can be solved using efficient linear programming techniques [22]. The solution of Eqn. (2) can be written as a decision matrix shown below:

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

Each row of X represents the policy for each node and defined as a *decision vector*$(dv)_i$. 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.

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 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* [22] whereby each node calculates its own dv. The distributed formulation can then be defined for each

node $i \in N$ as shown in Eqn.(4).

$$dv_i = \underset{x}{\operatorname{argmin}} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{4a}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = \gamma_i \tag{4b}$$

$$\sum_{i=1}^{n} x_{ji} + \gamma_{i0} \preceq \mu_{iCPU} \tag{4c}$$

$$x_{ij} \geq 0 \tag{4d}$$

This is similar to the *Gauss-Siedel* like method used by Meskar [23] for MCC. The algorithm basically communicates with its immediate neighbours to see what they can offer 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.(2) where each node i only tries to minimise the cost of its own objective function on the basis of information available on its neighbours. Similar to centralised problem, Eqn. (4) can be solved using linear programming techniques [22].

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. (2) and (4) 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:

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

where, L_i is the number of CPU jobs already in node i, T_i is the average processing time of each CPU Jobs and $\{\omega_k\}_1^3$ are weight factors. The significance of various components in Eqn.(5) can be changed using the weighting factor $\{\omega_k\}_1^3$. The algorithms can put more emphasis on one component than the other. For example, if the nodes are mains powered, ω_3 can be set to zero.

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 availabity for further external jobs and vice versa. This is also applicable for self-processing in the scheduling decision making.

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) [24], [25]. Results from [25] 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. (6a)):

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

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

The relationship (see Fig. 5) 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, 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. Once the retransmission factor is known, the communication cost between i and j, α_{ij} is defined as:

$$\alpha_{ij} = L_{iWS} T_{iWS} + D \times \frac{f+1}{BW_{ij}} + L_{jWR} T_{jWR} \tag{7}$$

where, BW_{ij} is the bandwidth between node i and j; D is the data size; f is the average retransmission times (see Eqn. 6a); α_{ij} is the communication cost; B_j is the remaining energy in node j (Joules),L_{iWS}, L_{jWR} are the number of jobs already in the WiFi send and receive queues of node i and j; T_{iWS}, T_{jWR} are expected WiFi sending and receiving time in i and j. Note that α_{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. (5) 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.

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.

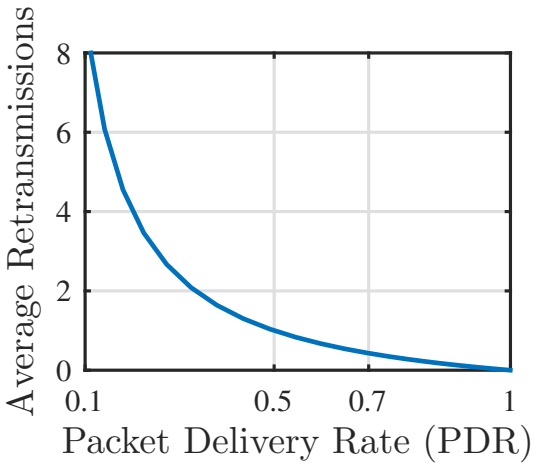


Fig. 5: Average no. of retransmissions required due to imperfect channel.

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. (2) and (4)). 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 (γ) in Eqn.(2) and (4) are non-stationary. The lowest sampling time of the simulator is $10ms$, hence the problems in Eqn.(2) and (4) 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. (2) every second which is every hundredth sampling step. Once solved, it sends the related policy 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.(2) and sends the corresponding policy dv back to each nodes. All other nodes are obliged to follow the decision made by the *server* and computes and offloads based on the policy dv_i until a new one is broadcast. However, different to Oracle, the cost of communication, as well as cost of executing the solver are taken into account.

Also, 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. Every minute a different *server* chosen which acts as the *server* and so on.

Also, there arises a question, how often the nodes need to broadcast their NSI and how often can they broadcast it

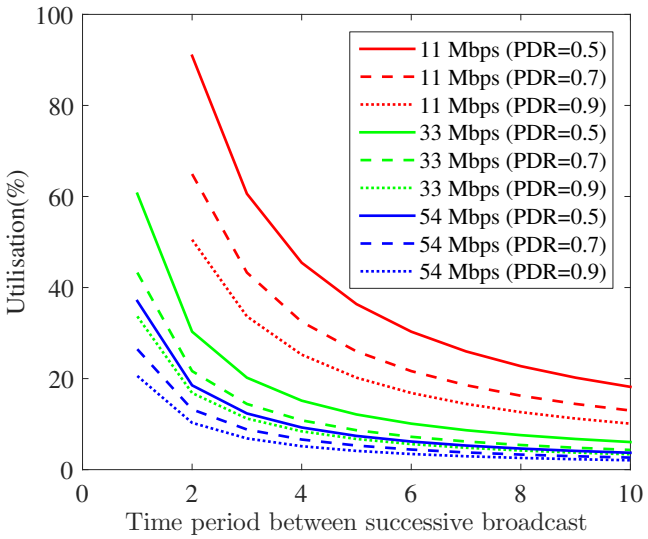


Fig. 6: Queue utilisation of *server* in proactive setting under various network conditions (Lower is better). Data size set at 1 Mb.

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 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:

$$\text{Arriving rate }, \lambda = \frac{n-1}{t} \tag{8}$$

$$\text{Worst Service rate }, \mu = \frac{\text{Data Rate} \times \text{worst PDR}}{\text{NSI size}} \tag{9}$$

$$\text{Utilization }, \rho = \frac{\lambda}{\mu} = \frac{(n-1) \times \text{ NSI size}}{t \times \text{Data Rate} \times PDR} \tag{10}$$

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

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

Based on the arriving rate and service rate, 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. (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. 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)

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), 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)

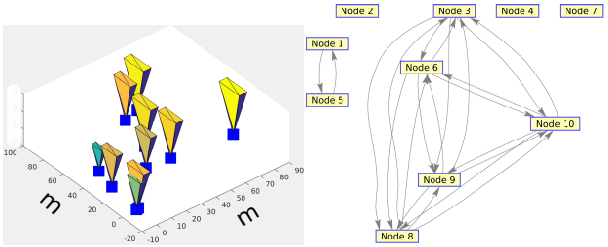
Algorithm 1 Reactive Distributed

if $\gamma_i + \gamma_i 0 \leq \mu_i$ **then**
  Set dv_i to not offload.
else
  if Request For Help (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 ≥ 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

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 [26]. 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. (4). 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.

5 Simulator Setup

In the 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 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 (more on this later). When it comes into the FOV of a sensor, it gets detected and once detected, the sensor has to identify the target. The major elements of our simulator relate to the algorithmic tasks, the sensor architecture, communication links and the targets.



(a) (b)

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 [27] 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

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 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) [27] 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 [27]. 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.

$$T_{\text{exec}} \propto \frac{C}{\text{Clock Frequency}} \tag{11}$$

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 [26] and our parameters are based on a Google Nexus I phone which was a Device Under Test (DUT) in [26]. If desired, the simulator can be calibrated for a different DUTs in a straightforward manner.

TABLE 4: CPU Parameters for the DUT (Google Nexus I) based on Jung et al. [26]

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

5.2.1 Application Processor (AP)

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

$$P^{\text{cpu}} = \beta^{\text{cpu}}_{\text{freq}} \times u + \beta^{\text{cpu}}_{\text{idle}}, \tag{12}$$

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 [26]. The utilisation 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 [28] 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:

$$u = \frac{\sum_{i=1}^{N} T_{\text{exec}i}}{T_{\text{Frame}}} + r \tag{13}$$

where N is the number of algorithms to be processed, T_{exec_i} is the execution time for i^{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. [29], when using the image sensor continuously, the energy consumption per frame of the image sensor can be modelled as:

$$E_{\text{camera}} = P_{\text{idle}} \times (T_{\text{frame}} - T_{\text{active}}) + P_{\text{active}} \times T_{\text{active}} \tag{14}$$

where, $T_{\text{frame}} = 1/\text{FPS}$ is time allocated for each frame, $T_{\text{active}} =$ 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. (14), 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.

$$p^{\text{wifi}} = \begin{cases} \beta_{\text{LT}} \times p + \beta_{\text{LT base}} & \text{if } p \leq P_{Th} \\ \beta_{\text{HT}} \times p + \beta_{\text{HT base}} & \text{if } p > P_{Th} \end{cases} \tag{15}$$

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 [26] (see Table 5).

TABLE 5: Image Sensor and WiFi Parameters

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

As per [26], 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. Although Wi-Fi consumes energy in scanning mode, it is ignored as connection between the sensors is the basis of this research.

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 Random Waypoint Model

In the Random Waypoint Model [30], targets spawn at random locations in the platform. It then moves around the platform just like a drone being flown first time by an amateur. 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. 9a).

5.3.2 SAIVT

A multi-camera scenario described in SAIVT Multi-Camera Surveillance Database [31] 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 instead of processing the images. According to the dataset [31], 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

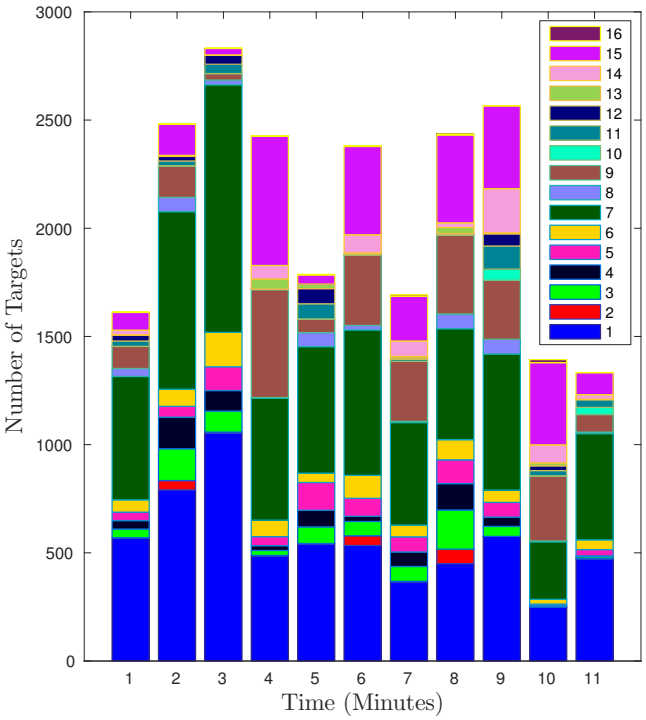


Fig. 8: Heterogeneous loading of cameras in multi-camera scenario. Each colour band represents target load on each camera. For example, the bottom and the top bands represents target arrivals in camera index 1 and 16 respectively. Majority of targets appear in camera indexed $1, 7$ and 15. (Best viewed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

TABLE 6: 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	$30, 60, 90$

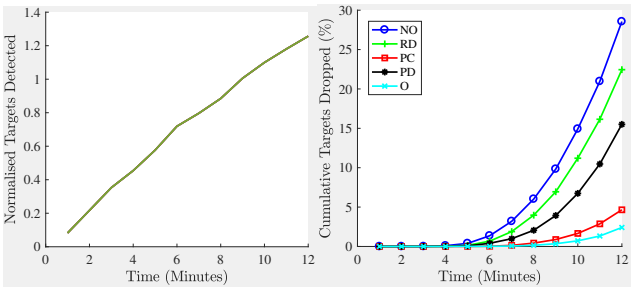
minutes of simulated time, on two sets of target data described in section (5.3). The total energy consumption for each sensor was estimated by summing power consumption of each component based on energy values from Eqn. (12), (14) and (15) in section (5.2). For each run, the simulator was initialised as per Algorithm (2). Each simulation was repeated for the various parameters to see if there is any effect on algorithm performance (see Table 6).

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 ∞ (i.e. no communication link).

6.1 Results for the standard configuration

Fig. 9a shows the average target detected across all the nodes and across all the trials, normalised by the total

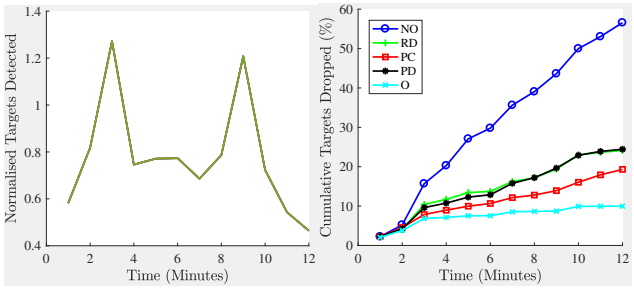


(a) (b)

Fig. 9: 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.

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. 9b 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 a later section.

Fig. 10a 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. 10a. In the remaining sub-sections, the performance is analysed with respect to the energy consumed as well as effect of environment and parameter selections.



(a) (b)

Fig. 10: 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.

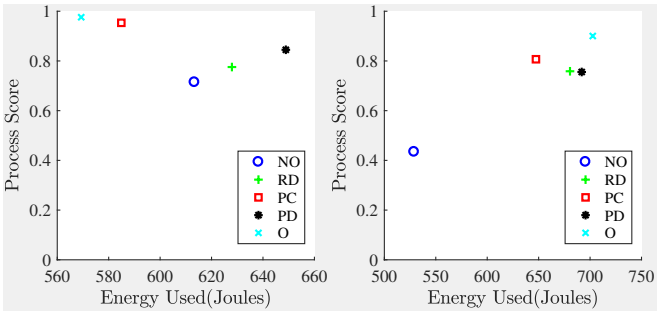
6.2 Process Score and Efficiency Score

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 [32] 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 7 and Fig. 11 for the standard configuration. For Fig. 11, 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 is justified and can help in selecting the right algorithm. This can be explained using an example, in Fig. 11a, 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 from Fig. 11 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 6.4). 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.

TABLE 7: 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	Algorithm	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



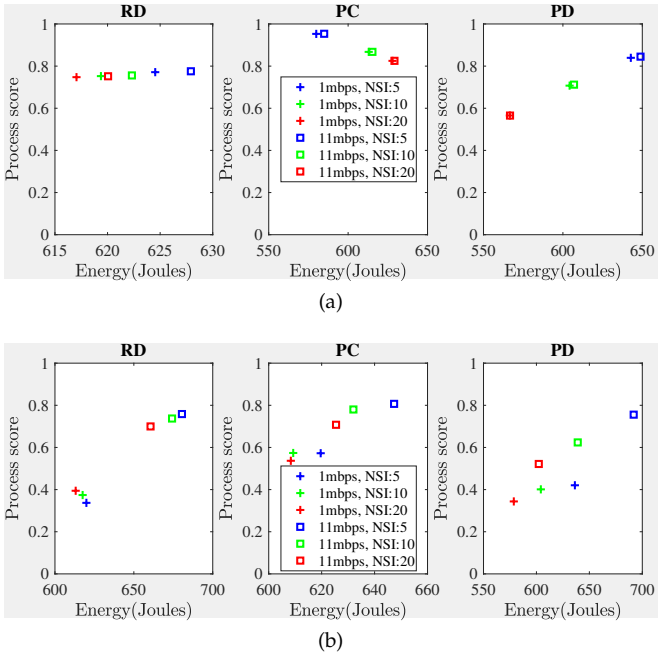
(a) (b)

Fig. 11: Efficiency Scores (a) RWP (b) SAIVT

6.3 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. 12a. 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. 12b). 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. (15), 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.

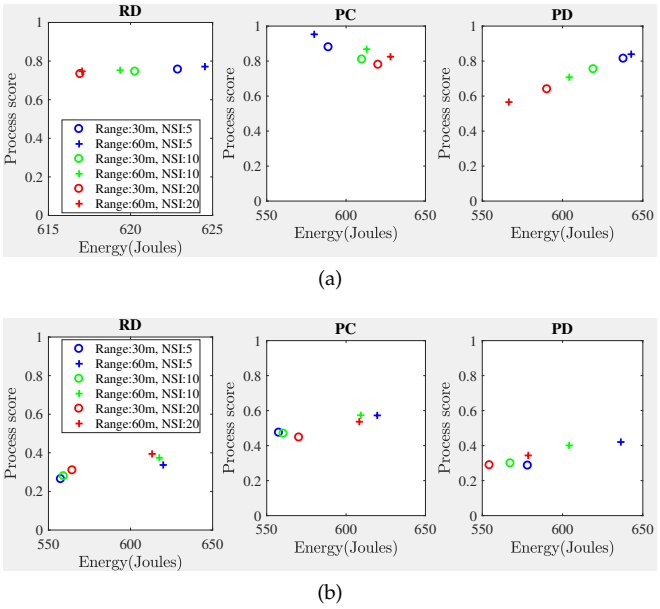
The performance of the proposed algorithms increased when the NSI exchanges were frequent (from once every 20 seconds to once every 5 seconds). This signifies the importance of having recent NSI about neighbouring nodes. Particularly, PD was highly dependant on the frequency of NSI exchange. When the frequency was low (once every 20



(a)

(b)

Fig. 12: 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. 13: 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**

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. 12, 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. 12a). However, upon further study, the energy usage was based more on CPU usage than on NSI exchanges.

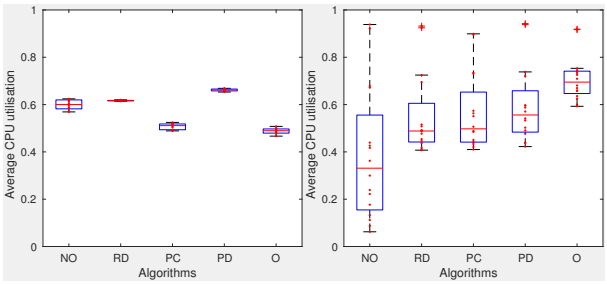
6.4 Effect of Communication range

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. 13. 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.

6.5 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. 14 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. 14b). 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



(a) (b)

Fig. 14: Average CPU utilisation across the nodes and NSI frequency (5, 10, and 30 seconds) (a) RWP (b) SAIVT

TABLE 8: 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

reduced CPU usage which suggests network lifetimes may be extended.

6.6 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 8. 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 case and often better. The O 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 may be due to the fact that it is able to consider 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 on the edge in absence of cloud and fog. 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 reinforces 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 other 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. Simulation data will be made available

References

[1] X. Wang and R. Zhao, "Person Re-Identification: System Design and Evaluation Overview," in *Pers. Re-Identification* (S. Gong, M. Cristani, S. Yan, and C. C. Loy, eds.), London: Springer London, 2014.

[2] X. Wang, "Face Identification," *Comput. Vis. A Ref. Guid.*, pp. 279–285, 2014.

[3] S. Sthapit, J. Thompson, J. R. Hopgood, and N. M. Robertson, "Distributed Implementation for Person Re-Identification," in *2015 Sens. Signal Process. Def.*, pp. 1–5, IEEE, sep 2015.

[4] S. Sthapit, J. R. Hopgood, N. M. Robertson, and J. Thompson, "Offloading to neighbouring nodes in smart camera network," *2016 24th Eur. Signal Process. Conf.*, 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," *Proc. 2015 Work. Mob. Big Data - Mobidata '15*, pp. 37–42, 2015.

[7] S. Sthapit, J. R. Hopgood, and J. Thompson, "Distributed Computational Load Balancing for Real-Time Applications," in *2017 25th Eur. Signal Process. Conf.*, 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 *Proc. 8th Int. Conf. Mob. Syst. Appl. Serv.*, 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 *Proc. Sixth Conf. Comput. Syst.*, 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 Commun. Surv. 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," *Mob. Networks Appl.*, vol. 18, no. 1, pp. 129–140, 2013.

[12] J. 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. Netw. Comput. Appl.*, vol. 48, pp. 99–117, 2015.

[13] 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.

[14] Y. Zhang, D. Niyato, and P. Wang, "Offloading in Mobile Cloudlet Systems with Intermittent Connectivity," *IEEE Trans. Mob. Comput.*, vol. 14, no. 12, pp. 2516–2529, 2015.

[15] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 974–983, 2015.

[16] 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, no. 2, pp. 421–449, 2016.

[17] 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," *IEEE Syst. J.*, vol. 9, no. 2, pp. 393–405, 2015.

[18] C. M. Sarathchandra Magurawalage, K. Yang, L. Hu, and J. Zhang, "Energy-efficient and network-aware offloading algorithm for mobile cloud computing," *Comput. Networks*, vol. 74, pp. 22–33, 2014.

[19] T. Truong-Huu, C.-K. Tham, and D. Niyato, "A Stochastic Workload Distribution Approach for an Ad Hoc Mobile Cloud," *2014 IEEE 6th Int. Conf. Cloud Comput. Technol. Sci.*, pp. 174–181, 2014.

[20] W. J. Stewart, *Probability, Markov chains, queues, and simulation : the mathematical basis of performance modeling*. Princeton (N.J.), Oxford: Princeton University Press, 2009.
[21] 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.
[22] S. Boyd and L. Vandenberghe, "Convex Set," *Convex Optim.*, pp. 21–60, 2004.
[23] 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.
[24] S. Biaz and S. Wu, "Rate Adaptation Algorithms for IEEE 802.11 Networks: A Survey and Comparison," *Proc. IEEE Symp. Comput. Commun.*, pp. 130–136, 2008.
[25] J. Zhang, K. Tan, J. Zhao, H. Wu, and Y. Zhang, "A practical SNR-guided rate adaptation," *Proc. - IEEE INFOCOM*, pp. 146–150, 2008.
[26] W. Jung, C. Kang, C. Yoon, D. D. Kim, and H. Cha, "DevScope: a nonintrusive and online power analysis tool for smartphone hardware components," *... Hardware/software ...*, pp. 353–362, 2012.
[27] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," Tech. Rep. January, 2008.
[28] W. Dargie, "A Stochastic Model for Estimating the Power Consumption of a Processor," *IEEE Trans. Comput.*, vol. 64, no. 5, pp. 1311–1322, 2015.
[29] R. Likamwa, B. Priyantha, M. Philipose, L. Zhong, and P. Bahl, "Energy characterization and optimization of image sensing toward continuous mobile vision," in *Proceeding 11th Annu. Int. Conf. Mob. Syst. Appl. Serv.*, (New York, NY, USA), pp. 69–82, ACM, 2013.
[30] F. Bai and A. Helmy, "A Survey of Mobility Models in Wireless Adhoc Networks," *Wirel. Ad Hoc Sens. Networks*, pp. 1–30, 2004.
[31] A. Bialkowski, S. Denman, S. Sridharan, C. Fookes, and P. Lucey, "A Database for Person Re-Identification in Multi-Camera Surveillance Networks," *2012 Int. Conf. Digit. Image Comput. Tech. Appl.*, pp. 1–8, dec 2012.
[32] H. Mao, S. Member, S. Yao, S. Member, T. Tang, and S. Member, "Towards Real-Time Object Detection on Embedded Systems," vol. 6750, no. 61373026, pp. 1–13, 2016.

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 co-leads 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.

Bibliography

[1] C. Kelly, "Ok, google, take a deep breath," *New York Times*, 2012. 1.1

[2] J. Aron, "How innovative is apple's new voice assistant, siri?," *New scientist*, vol. 212, no. 2836, 2011. 1.1

[3] D. Etherington, "Amazon echo is a $199 connected speaker packing an always-on siri-style assistant," November 2014. Accessed: 08/12/2017. 1.1

[4] A. Saini and T. Koyama, "Cleanup technologies following fukushima," *MRS Bulletin*, vol. 41, no. 12, p. 952954, 2016. 1.1

[5] M. Yamaguchi, "Swimming robot to probe damage at japan nuclear plant," June 2017. (document), 1.1, 1.1

[6] D. Science and T. Laboratory, "Ministry of defence unveils futuristic uniform design," September 2015. URL: https://www.gov.uk/government/news/mod-unveils-futuristic-uniform-design. (document), 1.3, 1.2

[7] I. Drury, "Soldier of the future: Maps beamed to his glasses, helmet camera sending images to comrades and sensors to monitor his health," September 2015. URL: http://www.dailymail.co.uk/news/article-3237525/Soldier-future-Maps-beamed-glasses-helmet-camera-sending-images-comrades-sensors-monitor-health.html. 1.3

[8] S. Martelli, D. Tosato, M. Cristani, and V. Murino, "Fpga-based pedestrian detection using array of covariance features," in *2011 Fifth ACM/IEEE International Conference on Distributed Smart Cameras*, pp. 1–6, Aug 2011. (document), 1.3

[9] 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. 1.5

[10] 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. 1.5

[11] 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. 1.5

[12] W. Hu, T. Tan, L. Wang, and S. Maybank, "A Survey on Visual Surveillance of Object Motion and Behaviors," *Trans. Sys. Man Cyber Part C*, vol. 34, pp. 334–352, aug 2004. (document), 2.2, 2.1

[13] L. Marcenaro, F. Oberti, G. L. Foresti, and C. S. Regazzoni, "Distributed architectures and logical-task decomposition in multimedia surveillance systems," *Proceedings of the IEEE*, vol. 89, pp. 1419–1440, oct 2001. 2.1, 2.1.1

[14] M. Valera and S. A. Velastin, "Intelligent distributed surveillance systems: a review," *IEE Proceedings - Vision, Image and Signal Processing*, vol. 152, pp. 192–204, apr 2005. 2.1

[15] S. A. Velastin, B. A. Boghossian, B. P. L. Lo, J. Sun, and M. A. Vicencio-Silva, "PRISMATICA: toward ambient intelligence in public transport environments," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 35, pp. 164–182, jan 2005. (document), 2.1, 2.1.1, 2.1.1, 2.4

[16] X. Wang, "Intelligent multi-camera video surveillance: A review," *Pattern Recognition Letters*, vol. 34, no. 1, pp. 3–19, 2013. 2.1

[17] M. Al Najjar, M. Ghantous, and M. Bayoumi, *Video Surveillance for Sensor Platforms*, vol. 114. 2014. (document), 2.3, 2.1.1, 2.1.1

[18] M. Richardson and S. Wallace, *Getting started with raspberry PI.* " O'Reilly Media, Inc.", 2012. 2.1.1

[19] NVIDIA Corporation, "NVIDIA Tegra K1: A new era in mobile computing," tech. rep., NVIDIA, 2014. 2.1.1

[20] D. H. Ballard and C. M. Brown, *Computer Vision.* Prentice Hall Professional Technical Reference, 1st ed., 1982. 2.2

[21] D. Marr, *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* New York, NY, USA: Henry Holt and Co., Inc., 1982. 2.2

[22] R. Gonzalez and R. Woods, *Digital Image Processing.* Pearson Education, 2009. 2.2.1

[23] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, (Washington, DC, USA), pp. 886–893, IEEE Computer Society, 2005. (document), 2.5, 2.2.2, 3

[24] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 971–987, jul 2002. 2.2.2, 2.3, 2.4.1

[25] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, nov 2004. 2.2.2, 2.3

[26] P. E. Forssen, "Maximally stable colour regions for recognition and matching," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, June 2007. 2.2.2, 2.4.2

[27] C. M. Bishop, *Pattern Recognition and Machine Learning*, vol. 4. 2006. 2.2.2, 2.4.4

[28] D. Gray, S. Brennan, and H. Tao, "Evaluating Appearance Models for Recognition, Reacquisition, and Tracking," in *10th IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS)*, 2007. (document), 2.3, 2.8

[29] S. Gong, M. Cristani, S. Yan, and C. C. Loy, *Person Re-Identification, Advance in Computer Vision and Pattern Recognition*. 2014. 2.3, 2.4

[30] A. Colombo, J. Orwell, and S. Velastin, "Colour constancy techniques for re-recognition of pedestrians from multiple surveillance cameras," in *Workshop on Multi-camera and Multi-modal Sensor Fusion Algorithms and Applications-M2SFA2 2008*, 2008. 2.3

[31] N. Dalal, *Finding people in images and videos*. PhD thesis, 2006. 2.3

[32] B. Prosser, W.-S. Zheng, S. Gong, and T. Xiang, "Person Re-Identification by Support Vector Ranking," in *Proceedings of the British Machine Vision Conference*, pp. 21.1—-21.11, BMVA Press, 2010. 2.3

[33] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960. 2.3.1

[34] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," *IEE Proceedings F - Radar and Signal Processing*, vol. 140, pp. 107–113, April 1993. 2.3.1

[35] L. Patino and J. Ferryman, "PETS 2014: Dataset and challenge," in *2014 11th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 355–360, aug 2014. (document), 2.3.1, 2.11

[36] 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. (document), 2.3.3, 4.2, 4.4

[37] R. Zhao, W. Ouyang, and X. Wang, "Learning Mid-level Filters for Person Re-identification," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 144–151, jun 2014. 2.4, 2.4.3

[38] M. Köstinger, M. Hirzer, P. Wohlhart, P. M. Roth, and H. Bischof, "Large scale metric learning from equivalence constraints," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2288–2295, jun 2012. 2.4, 2.1, 2.4.1, 2.4.1, 3.2.1, 3.2.2

[39] Z. Li, S. Chang, F. Liang, T. S. Huang, L. Cao, and J. R. Smith, "Learning Locally-Adaptive Decision Functions for Person Verification," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3610–3617, jun 2013. 2.4, 2.1, 2.4.1

[40] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon, "Information-theoretic Metric Learning," in *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, (New York, NY, USA), pp. 209–216, ACM, 2007. 2.4, 2.1, 2.4.1

[41] P. M. Roth, M. Hirzer, M. Köstinger, C. Beleznai, and H. Bischof, "Mahalanobis Distance Learning for Person Re-identification," in *Person Re-Identification*, pp. 247–267, Springer London, 2014. 2.4, 2.4.1

[42] M. Farenzena, L. Bazzani, A. Perina, V. Murino, and M. Cristani, "Person re-identification by symmetry-driven accumulation of local features," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2360–2367, jun 2010. 2.4, 2.4.2, 3.2.1

[43] H. Ma, C. Zeng, and C. X. Ling, "A Reliable People Counting System via Multiple Cameras," *ACM Trans. Intell. Syst. Technol.*, vol. 3, pp. 31:1—-31:22, feb 2012. 2.4, 2.1

[44] N. Gheissari, T. B. Sebastian, and R. Hartley, "Person Reidentification Using Spatiotemporal Appearance," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, pp. 1528–1535, 2006. 2.4, 2.1

[45] X. Wang and R. Zhao, *Person Re-identification: System Design and Evaluation Overview*, pp. 351–370. London: Springer London, 2014. 2.4, 3.2.1

[46] M. A. Saghafi, A. Hussain, H. B. Zaman, and M. H. M. Saad, "Review of Person Re-identification Techniques," *CoRR*, vol. abs/1511.02319, 2015. 2.4

[47] A. Bedagkar-Gala and S. K. Shah, "A survey of approaches and trends in person re-identification," *Image and Vision Computing*, vol. 32, no. 4, pp. 270–286, 2014. 2.4

[48] L. Bazzani, M. Cristani, and V. Murino, "Symmetry-driven accumulation of local features for human characterization and re-identification," *Computer Vision and Image Understanding*, vol. 117, no. 2, pp. 130–144, 2013. 2.1

[49] R. Zhao, W. Ouyang, and X. Wang, "Unsupervised Salience Learning for Person Re-identification," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3586–3593, jun 2013. 2.1, 2.4.3, 3.2.1, 3.2.1

[50] M. Guillaumin, J. Verbeek, and C. Schmid, "Is that you? Metric learning approaches for face identification," in *2009 IEEE 12th International Conference on Computer Vision*, pp. 498–505, sep 2009. 2.4.1

[51] R. Zhao, W. Ouyang, and X. Wang, "Person Re-identification by Salience Matching," in *2013 IEEE International Conference on Computer Vision*, pp. 2528–2535, dec 2013. (document), 2.4.3, 2.13

[52] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Deep Metric Learning for Person Re-identification," in *Proceedings of the 2014 22Nd International Conference on Pattern Recognition*, ICPR '14, (Washington, DC, USA), pp. 34–39, IEEE Computer Society, 2014. 2.4.4, 2.2

[53] W. Li, R. Zhao, T. Xiao, and X. Wang, "DeepReID: Deep Filter Pairing Neural Network for Person Re-identification," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 152–159, jun 2014. 2.4.4, 2.2

[54] E. Ahmed, M. Jones, and T. K. Marks, "An improved deep learning architecture for person re-identification," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3908–3916, jun 2015. (document), 2.4.4, 2.2, 2.14

[55] D. Cheng, Y. Gong, S. Zhou, J. Wang, and N. Zheng, "Person Re-Identification by Multi-Channel Parts-Based CNN With Improved Triplet Loss Function," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, jun 2016. 2.4.4, 2.2

[56] Z. Zheng, L. Zheng, and Y. Yang, "Unlabeled Samples Generated by {GAN} Improve the Person Re-identification Baseline in vitro," *CoRR*, vol. abs/1701.07717, 2017. 2.4.4

[57] W. Chen, X. Chen, J. Zhang, and K. Huang, "Beyond triplet loss: a deep quadruplet network for person re-identification," *CoRR*, vol. abs/1704.01719, 2017. 2.4.4, 2.2

[58] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *ArXiv e-prints*, June 2014. 2.4.4

[59] M. Geng, Y. Wang, T. Xiang, and Y. Tian, "Deep Transfer Learning for Person Re-identification," *CoRR*, vol. abs/1611.05244, 2016. 2.2, 2.4.4

[60] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian, "Scalable Person Re-identification: A Benchmark," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1116–1124, dec 2015. 2.4.4

[61] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. (document), 2.4.4, 2.4.4, 2.3

[62] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: {A} Tutorial and Survey," *CoRR*, vol. abs/1703.09039, 2017. 2.4.4

[63] M. S. L. B. Dong Seon Cheng, Marco Cristani and V. Murino, "Custom pictorial structures for re-identification," in *Proceedings of the British Machine Vision Conference*, pp. 68.1–68.11, BMVA Press, 2011. http://dx.doi.org/10.5244/C.25.68. 2.4.5

[64] W. Li, R. Zhao, and X. Wang, *Human Reidentification with Transferred Metric Learning*, pp. 31–44. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. 2.4.5

[65] W. Li and X. Wang, "Locally aligned feature transforms across views," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3594–3601, 2013. 2.4.5

[66] W. Li, R. Zhao, T. Xiao, and X. Wang, "DeepReID: Deep Filter Pairing Neural Network for Person Re-identification," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 152–159, jun 2014. 2.4.5

[67] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian, "Scalable person re-identification: A benchmark," in *Computer Vision, IEEE International Conference on*, 2015. 2.4.5

[68] L. Zheng, Z. Bie, Y. Sun, J. Wang, C. Su, S. Wang, and Q. Tian, *MARS: A Video Benchmark for Large-Scale Person Re-Identification*, pp. 868–884. Cham: Springer International Publishing, 2016. 2.4.5

[69] C.-K. Toh, *Wireless ATM and Ad-Hoc Networks: Protocols and Architectures*. Norwell, MA, USA: Kluwer Academic Publishers, 1996. 2.5

[70] B. Sklar, *Digital Communications: Fundamentals and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988. (document), 2.16

[71] I. Glover and P. M. Grant, *Digital communications*. Pearson Education, 2010. 2.5.1

[72] T. Rappaport, *Wireless Communications: Principles and Practice*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2nd ed., 2001. 2.5.1, 2.5.1

[73] M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communication Systems: Modeling, Methodology and Techniques*. Norwell, MA, USA: Kluwer Academic Publishers, 2nd ed., 2000. 2.5.1, 3

[74] 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. 2.5.1, 5.3.2

[75] K. Harika, D. P. Harini, K. V. N. Kumar, and K. Kondaiah, "A Distributed CSMA Algorithm for Maximizing Throughput in Wireless Networks," 2012. 2.5.1

[76] N. Ding, D. Wagner, X. Chen, A. Pathak, Y. C. Hu, and A. Rice, "Characterizing and Modeling the Impact of Wireless Signal Strength on Smartphone Battery Drain," *SIGMETRICS Perform. Eval. Rev.*, vol. 41, pp. 29–40, jun 2013. 2.5.1

[77] P. Cardieri, "Modeling interference in wireless ad hoc networks," *IEEE Communications Surveys Tutorials*, vol. 12, pp. 551–572, Fourth 2010. 1

[78] Z. Han, D. Niyato, W. Saad, T. Baar, and A. Hjrungnes, *Game Theory in Wireless and Communication Networks: Theory, Models, and Applications*. New York, NY, USA: Cambridge University Press, 1st ed., 2012. 1, 2.5.1

[79] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. 2006. 2.5.2

[80] A. Boukerche, B. Turgut, N. Aydin, M. Z. Ahmad, L. Bölöni, and D. Turgut, "Survey Paper: Routing Protocols in Ad Hoc Networks: A Survey," *Comput. Netw.*, vol. 55, pp. 3032–3080, sep 2011. 2.5.2, 2.5.2

[81] C. E. Jones, K. M. Sivalingam, P. Agrawal, and J. C. Chen, "A Survey of Energy Efficient Network Protocols for Wireless Networks," *Wireless Networks*, vol. 7, pp. 343–358, jul 2001. 2.5.2

[82] E. M. Royer and C.-K. Toh, "A review of current routing protocols for ad hoc mobile wireless networks," *IEEE Personal Communications*, vol. 6, pp. 46–55, apr 1999. 2.5.2

[83] E. Alotaibi and B. Mukherjee, "Survey Paper: A Survey on Routing Algorithms for Wireless Ad-Hoc and Mesh Networks," *Comput. Netw.*, vol. 56, pp. 940–965, feb 2012. 2.5.2

[84] S. Mao, "Fundamentals of communication networks," in *Cogn. Radio Commun. Networks*, pp. 229–250, Woodhead Publishing Limited, 2010. 2.5.2

[85] E. Yanmaz, M. Quaritsch, S. Yahyanejad, B. Rinner, H. Hellwagner, and C. Bettstetter, *Communication and Coordination for Drone Networks*, pp. 79–91. Cham: Springer International Publishing, 2017. 2.5.2

[86] G. He, "Destination-sequenced distance vector (dsdv) protocol," *Networking Laboratory, Helsinki University of Technology*, pp. 1–9, 2002. 2.5.2

[87] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized link state routing protocol for ad hoc networks," in *Proceedings. IEEE International Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century.*, pp. 62–68, 2001. 2.5.2

[88] R. Bellman, "On a routing problem," *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958. 2.5.2

[89] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993. 2.5.2, 2.7.2, 2.7.2, 4.4.1

[90] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing*

and Networking, MobiCom '98, (New York, NY, USA), pp. 85–97, ACM, 1998. 2.5.2, 2.5.3

[91] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pp. 90–100, Feb 1999. 2.5.2

[92] F. Bai and A. Helmy, "A Survey of Mobility Models in Wireless Adhoc Networks," *Wireless Ad Hoc and Sensor Networks*, pp. 1–30, 2004. 2.5.3, 10, 4.5.3

[93] N. Aschenbruck and E. Gerhards-Padilla, "A survey on mobility models for performance analysis in tactical mobile networks," *Journal of Telecommunications and Information Technology*, vol. 2, pp. 54—-61, 2008. 2.5.3

[94] R. R. Roy, *Handbook of Mobile Ad Hoc Networks for Mobility Models*. New York, NY, USA: Springer-Verlag New York, Inc., 1st ed., 2010. 2.5.3

[95] C. Bettstetter, H. Hartenstein, and X. Pérez-Costa, "Stochastic Properties of the Random Waypoint Mobility Model," *Wireless Networks*, vol. 10, pp. 555–567, sep 2004. (document), 10, 2.18

[96] W. Navidi and T. Camp, "Stationary distributions for the random waypoint mobility model," *IEEE Transactions on Mobile Computing*, vol. 3, pp. 99–108, jan 2004. 10

[97] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006. 2.6, 2.6.1, 2.6.1, 2.6.2

[98] W. J. Stewart, *Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling*. Princeton, NJ, USA: Princeton University Press, 2009. 2.6

[99] R. Diestel, *Graph theory {graduate texts in mathematics; 173}*. Springer-Verlag Berlin and Heidelberg GmbH & amp, 2000. 2.7.1

[100] J. R. Beveridge, D. Bolme, B. A. Draper, and M. Teixeira, "The CSU Face Identification Evaluation System," *Machine Vision and Applications*, vol. 16, pp. 128–138, feb 2005. 3.2.1

[101] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 3.3

[102] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A System for Large-scale Machine Learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, (Berkeley, CA, USA), pp. 265–283, USENIX Association, 2016. 3.3

[103] C. Johnson, *Radio Access Networks for UMTS: Principles and Practice*. Wiley Publishing, 2008. 3.4

[104] H. Holma and A. Toskala, *WCDMA for UMTS : HSPA evolution and LTE*. Chichester, West Sussex: Wiley, 5th edition. ed., 2010. 3.4

[105] Qualcomm Technologies, "Trepn Profiler Starter Edition User Guide," tech. rep., Qualcomm Technologies, 2014. 3.4.1

[106] 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. 4.1.1, 4.1.2, 5.2

[107] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, *Cuckoo: A Computation Offloading Framework for Smartphones*, pp. 59–79. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. 4.1.1

[108] R. Kemp, "Programming Frameworks for Distributed Smartphone Computing," 2014. 4.1.1

[109] N. Fernando, S. W. Loke, and W. Rahayu, "Dynamic mobile cloud computing: Ad hoc and opportunistic job sharing," in *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pp. 281–286, Dec 2011. 4.1.2

[110] 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. 4.1.2

[111] M. Whaiduzzaman, A. Naveed, and A. Gani, "MobiCoRE: Mobile Device based Cloudlet Resource Enhancement for Optimal Task Response," *IEEE Transactions on Services Computing*, vol. X, no. X, pp. 1–1, 2016. 4.1.2

[112] T. Truong-Huu, C.-K. Tham, and D. Niyato, "To Offload or to Wait: An Opportunistic Offloading Algorithm for Parallel Tasks in a Mobile Cloud," *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pp. 182–189, 2014. 4.1.2

[113] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, 2013. 4.1.2

[114] 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. 4.1.2

[115] 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. 4.1.2

[116] R. Roman, J. Lopez, and M. Mambo, "Mobile Edge Computing, Fog et al.: A Survey and Analysis of Security Threats and Challenges," *Future Generation Computer Systems*, p. 28, 2016. 4.1.2

[117] 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. 4.1.2

[118] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, pp. 1628–1656, thirdquarter 2017. 4.1.2

[119] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "Mobile edge computing: Survey and research outlook," *CoRR*, vol. abs/1701.01090, 2017. 4.1.2

[120] 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. 4.1.2, 5.2

[121] P. Natarajan, P. K. Atrey, and M. Kankanhalli, "Multi-Camera Coordination and Control in Surveillance Systems: A Survey," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 11, pp. 57:1—-57:30, jun 2015. 4.1.2

[122] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction," in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 2, pp. 28–31 Vol.2, aug 2004. 4.2.1

[123] Y. Jiang, "A Survey of Task Allocation and Load Balancing in Distributed Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, pp. 585–599, feb 2016. 4.2.2

[124] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, pp. 483–485, ACM, 1967. 4.2.2

[125] M. Woolridge, *Introduction to Multiagent Systems*. New York, NY, USA: John Wiley & Sons, Inc., 2001. 4.2.4

[126] 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. 4.3.1

[127] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An analysis of TCP processing overhead," *IEEE Communications Magazine*, vol. 27, pp. 23–29, jun 1989. 4.3.1

[128] M. Caramia and P. DellOlmo, "Multi-objective Optimization," in *Multi-objective Management in Freight Logistics: Increasing Capacity, Service Level and Safety with Optimization Algorithms*, pp. 11–37, Springer London, 2008. 4.3.2, 4.4.1

[129] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004. 4.4.1, 5.5.1, 5.5.2, 5.5.2

[130] T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*. Springer Publishing Company, Incorporated, 2nd ed., 2011. 4.5

[131] G. Carneiro, "Ns-3: Network simulator 3," in *UTM Lab Meeting April*, vol. 20, 2010. 4.5

[132] X. Chang, "Network simulations with OPNET," *Simulation Conference Proceedings, 1999 Winter*, vol. 1, pp. 307–314 vol.1, 1999. 4.5

[133] J. C. SanMiguel and A. Cavallaro, "Networked Computer Vision: The Importance of a Holistic Simulator," *Computer*, vol. 50, no. 7, pp. 35–43, 2017. 4.5

[134] A. Boulis, "Castalia: Revealing pitfalls in designing distributed algorithms in wsn," in *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, SenSys '07, (New York, NY, USA), pp. 407–408, ACM, 2007. 4.5

[135] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta, "Evaluating the Effectiveness of Model-based Power Characterization," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, (Berkeley, CA, USA), p. 12, USENIX Association, 2011. 4.5.1

[136] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES/ISSS '10, (New York, NY, USA), pp. 105–114, ACM, 2010. 4.5.1

[137] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the Energy Spent Inside My App?: Fine Grained Energy Accounting on Smartphones with Eprof," in *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, (New York, NY, USA), pp. 29–42, ACM, 2012. 4.5.1

[138] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "AppScope: Application Energy Metering Framework for Android Smartphone Using Kernel Activity Monitoring," in *Presented as part of the 2012 {USENIX} Annual Technical Conference ({USENIX} {ATC} 12)*, (Boston, MA), pp. 387–400, USENIX, 2012. 4.5.1

[139] K. Kim, D. Shin, Q. Xie, Y. Wang, M. Pedram, and N. Chang, "FEPMA: Fine-grained event-driven power meter for android smartphones based on device driver layer event monitoring," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1–6, mar 2014. 4.5.1

[140] 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. 4.5.1, 4.5.1, 4.5.1, 5.6.4

[141] M. Lauridsen, *Studies on Mobile Terminal Energy Consumption for LTE and Future 5G*. PhD thesis, Aalborg University, 2014. 4.5.1

[142] 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. (document), 4.5.1, 4.7

[143] W. Dargie, "A Stochastic Model for Estimating the Power Consumption of a Processor," *IEEE Transactions on Computers*, vol. 64, pp. 1311–1322, may 2015. 4.5.1

[144] C. W. Johnson, *Radio Access Networks for UMTS: Principles and Practice*. John Wiley and Sons, 2011. 4.5.1, 4.5.1

[145] T. Guérout, T. Monteil, G. Da Costa, R. Neves Calheiros, R. Buyya, and M. Alexandru, "Energy-aware simulation with DVFS," *Simulation Modelling Practice and Theory*, vol. 39, pp. 76–91, 2013. 4.5.2

[146] 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. 4.5.4

[147] N. A. Pantazis, S. A. Nikolidakis, and D. D. Vergados, "Energy-Efficient Routing Protocols in Wireless Sensor Networks: A Survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 2, pp. 551–591, 2013. 5.1

[148] K. Ashton, "That 'internet of things' thing," *RFID Journal*, 2009. 5.1

[149] C. R. Schoenberger and B. Upbin, "The internet of things," *Forbes Magazine*, vol. 169, no. 6, pp. 155–160, 2002. 5.1

[150] 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. 5.2

[151] Y. Zhang, D. Niyato, and P. Wang, "Offloading in Mobile Cloudlet Systems with Intermittent Connectivity," *IEEE Transactions on Mobile Computing*, vol. 14, pp. 2516–2529, dec 2015. 5.2

[152] X. Chen, "Decentralized Computation Offloading Game for Mobile Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, pp. 974–983, apr 2015. 5.2

[153] 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. 5.2

[154] 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," *IEEE Systems Journal*, vol. 9, pp. 393–405, jun 2015. 5.2

[155] 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. 5.2

[156] 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. 5.2

[157] 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. 5.3.2

[158] J. E. Freund, *Mathematical Statistics*. Prentice-Hall, 5th edition ed., 1992. 5.3.2

[159] J. F. C. Kingman, "On queues in heavy traffic," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 24, no. 2, pp. 383–392, 1962. 5.4

[160] R. Williams, "On the approximation of queueing networks in heavy traffic," *Stochastic Networks: Theory and Applications*, no. 4, pp. 35–56, 1996. 5.4

[161] E. Meskar, T. D. Todd, D. Zhao, and G. Karakostas, "Energy Aware Offloading for Competing Users on a Shared Communication Channel," *IEEE Transactions on Mobile Computing*, vol. 16, pp. 87–96, jan 2017. 5.5.2

[162] 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. (document), 5.11, 5.7.2