ENERGY-AWARE MOBILE EDGE COMPUTING FOR LOW-LATENCY VISUAL DATA PROCESSING

A Thesis presented to
the Faculty of the Graduate School
at the University of Missouri

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

---

by
HUY TRINH
Dr. Prasad Calyam, Thesis Supervisor
DECEMBER 2017

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

ENERGY-AWARE MOBILE EDGE COMPUTING
FOR LOW-LATENCY VISUAL DATA PROCESSING

presented by Huy Trinh,

a candidate for the degree of Master of Science and hereby certify that, in their opinion,

it is worthy of acceptance.

Dr. Prasad Calyam

Dr. Kannappan Palaniappan

Dr. Timothy Middelkoop

ACKNOWLEDGMENTS

I would like to thank Prof. Prasad Calyam immensely for being my advisor and mentor who brought out the best in me during my thesis research. I also thank Prof. Calyam, for giving me the opportunity to be part of his numerous, challenging research projects. I am extremely grateful for his constant guidance, insight and encouragement through the entire course of my research. I would like to express my gratitude to Prof. Kannappan Palaniappan and Prof. Timothy Middelkoop for their interest in my research and consenting to be part of my thesis committee. I would like to acknowledge Dimitrii Chemodanov - my lab mate from VIMAN Lab who collaborated with me at various stages of my thesis. He helped discuss my research and providing helpful advice for my research work. I would like to thank Shizeng Yao, Qing Lei, Bo Zhang, Fan Gao who collaborate with me in Cloud Computing 1 project. My final words go to my family. I want to thank my parents, whose love and guidance is with me in whatever I pursue.

Contents

List of Tables

List of Figures

Abstract

New paradigms such as Mobile Edge Computing (MEC) are becoming feasible for use in e.g., real-time decision-making during disaster incident response to handle the data deluge occurring in the network edge. However, MEC deployments today lack flexible IoT device data handling such as e.g., handling user preferences for real-time versus energy-efficient processing. Moreover, MEC can also benefit from a policy-based edge routing to handle sustained performance levels with efficient energy consumption.

In this thesis, we study the potential of MEC to address application issues related to energy management on constrained IoT devices with limited power sources, while also providing low-latency processing of visual data being generated at high resolutions. Using a facial recognition application that is important in disaster incident response scenarios, we propose a novel 'offload decision-making' algorithm that analyzes the tradeoffs in computing policies to offload visual data processing (i.e., to an edge cloud or a core cloud) at low-to-high workloads. This algorithm also analyzes the impact on energy consumption in the decision-making under different visual data consumption requirements (i.e., users with thick clients or thin clients). To address the processing-throughput versus energy-efficiency tradeoffs, we propose a 'Sustainable Policy-based Intelligence-Driven Edge Routing' (SPIDER) algorithm that uses machine learning within Mobile Ad hoc Networks (MANETs). This algorithm improves the geographic routing baseline performance (i.e., minimizes impact of local minima) for performance sustainability, and enables easy/flexible policy specification. We evaluate our proposed algorithms by conducting experiments on a realistic edge and core cloud testbed, and recreate disaster scenes of tornado damages (occurred in Joplin, MO in

2011) within simulations. From our empirical results obtained from experiments with a facial recognition application in the GENI Cloud testbed, we show how MEC can provide flexibility to users who desire energy conservation over low-latency or vice versa in the visual data processing. Our NS-3 based simulation results show that our routing approach is more sustainable in terms of throughput, more energy-efficient and flexible than existing solutions to handle diverse user preferences under high node mobility and severe node failure conditions.

Chapter 1

Introduction

1.1 How Mobile Edge Computing and IoT Support Disaster Incident Response

The Internet of Things (IoT) is becoming increasingly relevant for innovations in smart city applications such as manufacturing and public safety. Mobile devices, wearable smart devices and sensors are being connected with diverse network connectivity options (e.g., austere infrastructure, Gigabit network speeds at the network edge), and applications can benefit from the insights in the data from these IoT devices. Especially for applications such as disaster incident response or law enforcement, visual data (e.g., high-resolution images, video clips) from IoT devices needs to be processed in real-time. For instance, the streaming application run 30 frame per seconds, but face detection only require first frame in the sequence [1]. Thus, the visual data processing application can reduce to 15 frame per seconds or approximately 0.1 second per image and save bandwidth as well as conserve real-time feature. Relevant insights from the visual data can help incident commanders to quickly analyze scenes and deploy resources (e.g., paramedics, ambulances, medical supplies) [2]. Through convergence with cloud computing technologies, IoT-based application data can be handled at large scale from multiple network edge sites with on-demand computation capabilities.

1.2 The Need of Flexible Policy-based in MEC Architecture

However, it is not always reasonable to assume that fully functional computing/networking infrastructure, and unlimited power sources exist to handle the visual data processing needs. In natural disaster situations involving earthquakes, hurricanes, or man-made disasters involving terrorism, edge infrastructure may be lost and computation for disaster incident response decision-making might require relying on constrained mobile devices in terms of computing, networking or power resources. One important IoT-based application we can envisage that is useful involves facial recognition technology, which provides fast and accurate identification when high-resolution image data, and high-performance computing/networking exist to match against a large/distributed database of images. The identification can help find 'lost persons' or identify 'bad actors' [3] in disaster scenes, if achieved in real-time and through processing of a high volume of images at the network edge on a limited power budget.

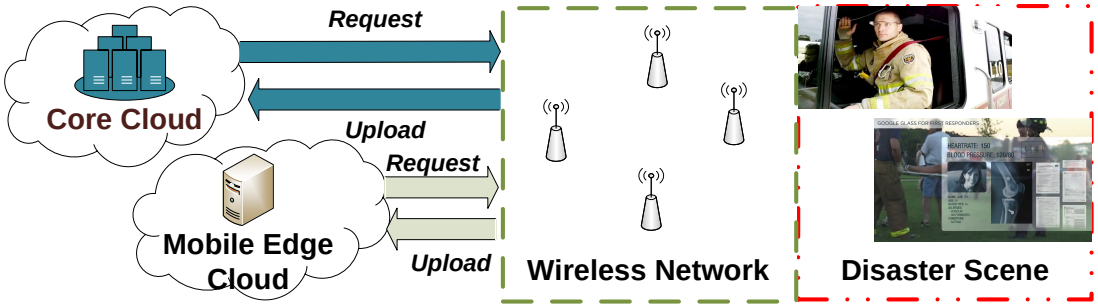


Figure 1.1: Illustration of disaster scene related visual data processing by use of wireless network, mobile edge and core cloud resources to upload images and request processed images using sophisticated computer vision algorithms.

Figure 1.1 shows an illustration of how new paradigms of Mobile Edge Computing (MEC) [4, 5] are emerging that allow for upload of raw images and download requests

of processed images in the exemplar facial recognition application context. MEC architectures allow for distributed computing in Radio Access Networks (RANs) by having cellular operators to cooperate application developers and content providers. Using MEC, we can augment critical infrastructure by having the cloud computing resources more distributed and accessible close to the wireless network-edge. For instance, it allows for a base station infrastructure or 'cloudlets' to handle computation requests from mobile devices that are in the geographic vicinity [6, 7]. According to [8], the offloading can save 25 times power consumption and 3 times processing times than processing on local device. Also, the mobile devices computation energy dominate about 75% during image processing and offloading. Thus, MEC provides options to offload computation tasks from IoT devices to address application issues related to energy management on constrained IoT devices with limited power sources, while also providing low-latency processing of visual data using sophisticated computer vision algorithms. However, there is a need for better understanding on the MEC paradigm potential in terms of its benefits or limitations when edge clouds are used with a core cloud that may have: (a) undesirably long round-trip times, (b) intermittent connectivity, or (c) excessive congestion, as in the case of austere or adverse network edge environments.

Moreover, MEC can also benefit from a policy-based edge routing to best balance between sustainability (throughput, end-to-end latency etc.) and energy consumption (network lifetime). However, in case of disaster scenario, the location of mobility nodes is changed frequently in Mobile Ad hoc Network (MANET) and static nodes consist of the absence of fixed infrastructure due to lack of power. Hence, these network nodes can cause often unpredictably topology changes [9] and it becomes hard to guarantee service continuity as well as maintain routing table in MEC networks. Many proto-

cols [10, 11] (e.g. stateful) have been found to achieve efficient routing in MANETs. These approaches are different while deploying new route and/or adjust existing routes, when events occur (e.g., high mobility or failure node). By maintain node positions in database and updating frequently, this causes huge overhead and energy wasting on power constrained IoT devices. Other researchers have found protocols [12, 13] (e.g. stateless) to route without maintaining routing table but degrade performance due to local minima. This can cause infinite loops in routing work and drain energy faster on IoT devices. In consequence, there is a need for stateless and energy-aware geographic routing with flexible policy and improved performance.

In this thesis, we aim to study the potential of the MEC paradigm by using the context of a facial recognition application in a disaster incident response scenario. Our goal is to adopt MEC within the facial recognition application framework and analyze the tradeoffs in computing policies that offload visual data processing (i.e., to an edge or a core cloud) at low-to-high workloads, and their impact on energy consumption under different visual data consumption requirements.

As part of thesis contributions, we particularly consider visual data consumption for users with *thin client* or *thick client* configurations; thin client configuration at a user assumes all of the processed images are stored and viewed at a remote cloud resource, whereas thick client configuration assumes processed images are downloaded and further post-processed at the mobile user device level. When available, we assume the core cloud has the option to provide multiple compute instances which can help in *parallel* processing of visual data workloads, versus having limited edge cloud resources that process the workloads in a *sequential* manner. Further, we consider cases where compression is used in the image transfers, which could save bandwidth con-

sumption in austere networks, but increases the energy consumption that could have a negative impact on the power-constrained IoT device or edge cloud side with limited power sources.

To provide a flexible option for IoT-based applications to decide whether to offload the visual data processing to an edge cloud or a core cloud for the above user requirement cases, we present a novel 'decision-making algorithm'. Our algorithm handles cases where a hard real-time processing need exists or a varying scale of visual data processing workload needs to be handled at the network-edge, while meeting user requirements that are energy conscious or demand fast processing.

We evaluate our energy-aware and low-latency MEC framework featuring the facial recognition application and our decision-making algorithm with experiments in a realistic edge and core cloud testbed. For the edge cloud, we use a campus server, and we use the GENI Cloud resource [14] for the core cloud. We leverage the Android-based PowerTutor utility [15] to profile and estimate energy consumption (Metric: Joules) of our facial recognition application that is based on OpenCV [16] within the testbed. Our experiment results show how MEC can provide flexibility to users who desire energy conservation over low-latency (Metric: Processing Time) or vice versa in visual IoT-based application data processing. We compare cases where using thin client or thick client configurations are more effective at low-to-high visual data processing workloads, and how offloading policies could affect the energy efficiency or low latency user requirements.

To achieve energy management on constrained IoT networks in geographic routing, we propose a Sustainable Policy-based Intelligence-Driven Edge Routing (SPIDER) algorithm that utilizes machine learning techniques to learn the geo-information about

existing physical obstacle from satellite images. In addition to adopt geographic coordinates (through GPS system), the knowledge is then used to improve the geographic routing which boosts baseline performance (i.e., avoid the impact of local minima). Also, we particularly consider edge network routing with easy policy specification such as performance sustainability versus network-lifetime and able to handle real-time situations.

We evaluate our SPIDER algorithm with recreating disaster scenes of tornado damages happens in Joplin areas, Missouri, USA in 2011. For the simulator, we use NS-3 (Network Simulator) to generate the disaster scenes with mobility and severe failures. We leverage the average remaining power which indicate network lifetime (Metric: Joules) of our UDP streaming application within the simulation. Our simulation results show how intelligent and edge routing can provide flexibility to users who desire energy conservation over performance sustainability (Metric: Processing Time and throughput) or vice versa in MEC.

1.3 Thesis Outline

The remainder of thesis is organized as follows: Chapter 2 reviews related work. In Chapter 3, we present our facial recognition application and a MEC framework for studying computation offloading policies to balance tradeoffs in energy efficiency and low-latency processing of low-to-high scale workloads from IoT devices. Chapter 4, we present SPIDER algorithm with machine learning to improve geographic routing baseline performance. We show realistic GENI Cloud testbed experiments, recreate disaster scenarios in simulation and performance evaluation are described in Chapter 6. Chapter 7 with future works and chapter 8 concludes the thesis work.

Chapter 2

Related Works

2.1 Computation Offloading Decision-Making.

Existing literature on computation offloading can be classified under two categories of work. First set of works such as [17, 18] consider the concept of "program partition", which involves offloading parts of a given processing task onto edge servers, and other parts of the task run on user devices. Specifically, they propose offline heuristic algorithms to support a large-scale mobile application and thereby reduce the completion time for all application users. A second set of works, such as [19, 20] consider a "migration" strategy that offloads the entire application onto an edge server. Specifically, the authors in [19] create a device classification for prioritizing computation that is based on the channel and base station resource allocation status. In [20], the authors use a Markov decision process to dynamically offload computation within services. If offloading is not a viable option, authors in works such as [21, 22] propose "load shedding", a prevalent data-stream management technique. Load shredding involves automatically either dropping or adapting the quality of packets on the edge device. Our work differs from existing works due to the energy-awareness and low-latency user requirements handling we address that flexibly allows visual data processing to occur either at the edge cloud or in the core cloud depending on the tradeoffs involved.

2.2 Visual Data Consumption.

To display visual data from a remote system, it is common to use either thin client or thick client solutions. A thin client [23] can typically run on local computer hardware (e.g., keyboard, mouse, display) that is able to remotely connect to a remote desktop that is either cloud-hosted or on a remote server. The computation burden in this case will reside on the server side, and screen scrapes are sent to the client. A thick client, on the other hand, can be assumed to be a fully functional computer or device that possess computing resources that are significant for post-processing visual data based on user drill-down or zoom in/out. According to [24], a stateless thick client might still require periodic connection and computation assistance from the cloud or a remote server. Regardless, user satisfaction in terms of image rendering quality and interaction depends on the session latency that depends on the network bandwidth and computational resources at the client/server sides. The authors in [25] found from real-world measurements that even with good bandwidth of 100 Mbps, the latency in thin clients still falls in range of 33-100 ms across different cities. Moreover, they recommend the use of "cloudlet" or "Mobile Edge Computing" architectures as a suitable solution to lower end-to-end latency. Our work builds upon this recommendation in our visual data processing workflow that is part of the MEC architecture based facial recognition application.

2.3 Geographic Routing for MANET.

When geographic routing is used, all nodes forward packets to the closest to the destination neighbors until packets are delivered. One of the stateful geographic routing

framework in MANETs is Destination Sequenced Distance Vector (DSDV) [26]. When utilize DSDV, each node updates a routing table of the next hop, number of hops periodically. Clearly, the accuracy of this approach depends on the frequency of flooding and the velocity of nodes. Due to limited energy of the IoT devices, energy-awareness is an important characteristic for geographic routing protocols. The technique such as [27] updating the energy values in the nodes routing table. Flooding happens occasionally to collect the energy information in the network. The destination nodes achieve energy-efficient by keeping track of the residual energy of their network nodes' battery levels. All aforementioned approaches need to maintain some knowledge about the network reflected in routing tables of nodes resulting in a network overhead with routing protocols. The well known energy-aware stateless routing proposed in [28] measures the distance between source node and sink node to estimate the energy consumption for sent packets. This approach is proven to provide low overhead while giving some energy-efficiency benefits over traditional stateful analogues.

To further improve energy-efficiency more recent works on clustering routing such as in [29] utilize users' mobility for selecting cluster heads which are then used to route packtes toward the Gateway. However, such cluster heads take some time to scan neighboring nodes' signals and store their information to form clusters. Thus, in case of severe node failures and mobility such approach may not provide a sufficient performnace sustainability. Similarly, the common cognitive routing approach [30, 31] uses techniques such as Quality of Information (QoI) and traffic awareness to optimaze energy-efficiency of the data transmission. However, such intelligence in a neighbor selection may again affect the performance sustainability of the whole network.

In this thesis, we propose the Sustainable Policy-based Intelligence-Driven Edge

Routing (SPIDER) approach that benefits from the physical obstacle knowledge derived from the satellite imagery by using deep learning-based detectors [32, 33] available at the edge. SPIDER builds upon our previous work [34] that theoretically guarantees avoidance of a local minimum as well as the shortest path approximation. Hence, this approach shows practiacal performance improvements over the recent stateless geographic routing algorithm which uses a notion of pressure forwarding for a local minimum recovery [35]. Thus, SPIDER demonstrates significant sustainability improvements over existing geographic routing solutions in MANETs by better coping with high mobility and severe node failures. We remark that such performance sustainability is crucial for our facial recognition application where we need a consistent data transfer rate. However, SPIDER has been also extended to take into account *flexibility* needs to best leverage trade-offs between the energy-efficiency and the sustainable performance by specifying corresponding policies that differentiates it from the above mentioned stateful and stateless geographic routing protocols.

Chapter 3

Energy-aware and Low-latency MEC Framework

In this section, we first describe the facial recognition technology and our application framework implementation that is important in disaster incident response when used by incident commanders and first responders. Following this, we detail our computation offloading decision-making algorithm that can handle scalable workloads and energy constraints of IoT devices that use our facial recognition application.

3.1 Application Background and Implementation

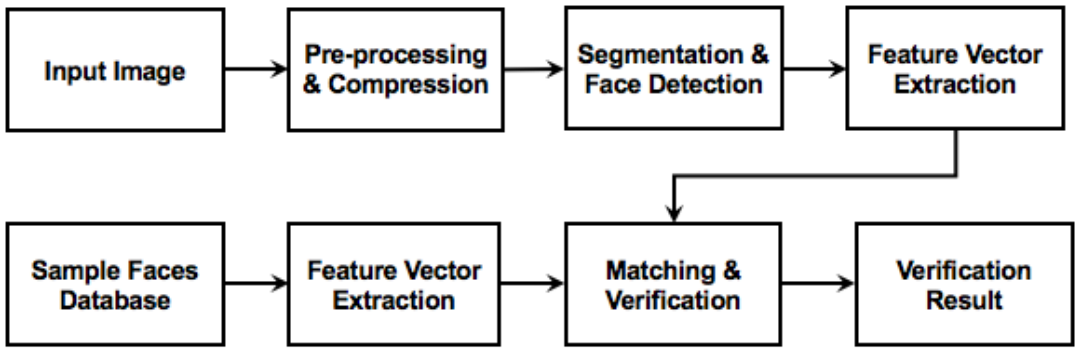


Figure 3.1: Overview of steps in facial recognition for target identification.

Facial recognition technology when used in an application on a mobile device can help in identifying or verifying a person's identity whose digital image is collected from a local camera/video source. The facial recognition process we use in our work has several steps as shown in Figure 3.1 that involve the digital image at the client side

and a larger image sample dataset at the server side.

To initially detect a human face for a given database of images within a small amount of time (i.e., with low latency), we perform a pre-processing step during which we compress all the input images. After compression, the face region in each given image is detected and segmented using eigenfaces techniques described in [36]. With the segmented face information, we apply feature extraction using a Histograms-of-Oriented-Gradients (HoG) [37] feature detector and save all of the extracted features into a target feature vector. Once we have the target feature vector information, we perform matching between target feature vector and multiple feature vectors of sample candidates in the sample image database such that each candidate in the database will have a matching score. In our implementation, this step could consume long processing times, especially when there are a large number of candidate images in the database. After calculating all the individual matching scores, we choose the candidate with the highest matching score in the final verification result.

Figure 3.2 shows the graphical user interface of our application implementation that is developed for an Android device using the Java programming language. The facial recognition process described above has been implemented using OpenCV [16] for image management and using Python scripts that utilize Dlib [38], an open source library. To use this interface and obtain, for instance, the name of the matched image, a user can choose different computation and image transfer polices as shown in the right half of Figure 3.2 such as: compression on device or server, thin or thick client, serial or parallel processing. The resulting image of the target identification along with processing time consumption can be obtained from the server side as shown in the left half of Figure 3.2 for single or multiple image uploads from one or more IoT devices

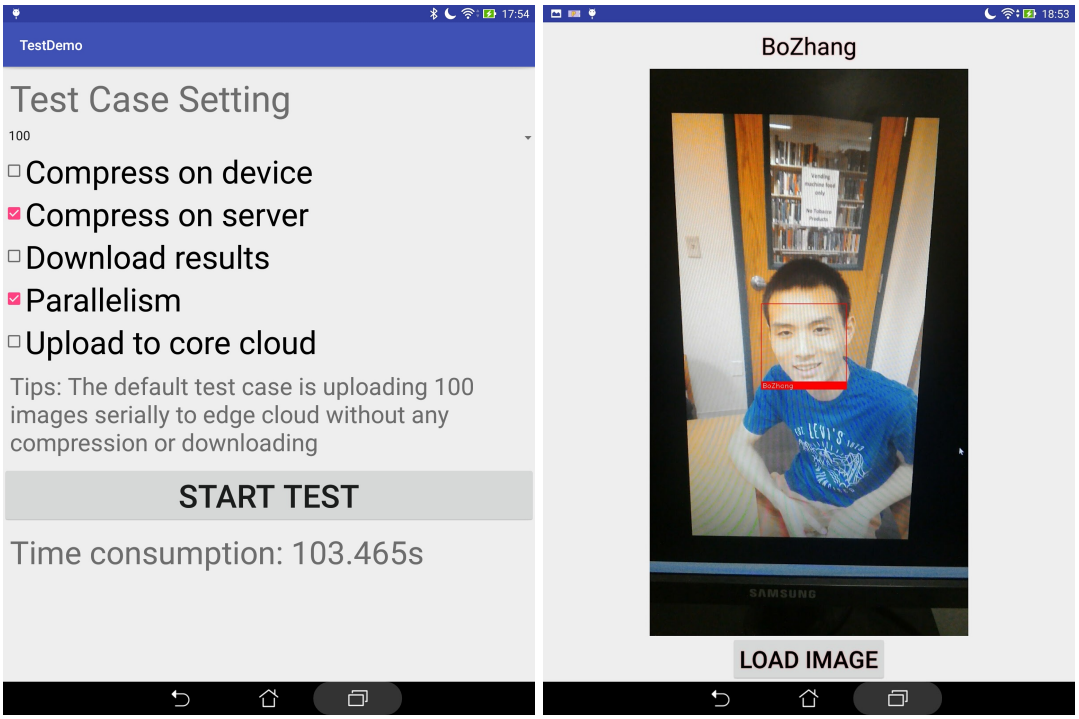


Figure 3.2: Application GUI - Left Image: test options for user to select, Right Image: received result from the server side.

simultaneously.

3.2 Computation Offloading Decision-Making Approach

The thin client or thick client application simply sends the data from the mobile device to a cloud server to achieve better results in low-latency processing and the related energy consumption. However, the decision between offloading computation to the edge or core cloud depends on the user requirement and workload scale. Authors in [6] show that the edge cloud improves response time from 200ms to 80ms and energy consumption reduced by 30%-40%. However, the core cloud is helpful because of unlimited resources and parallel instances to speedup processing. Therefore, we propose an algorithm to classify the scenario with the user's choice to help choose the best visual data processing decision.

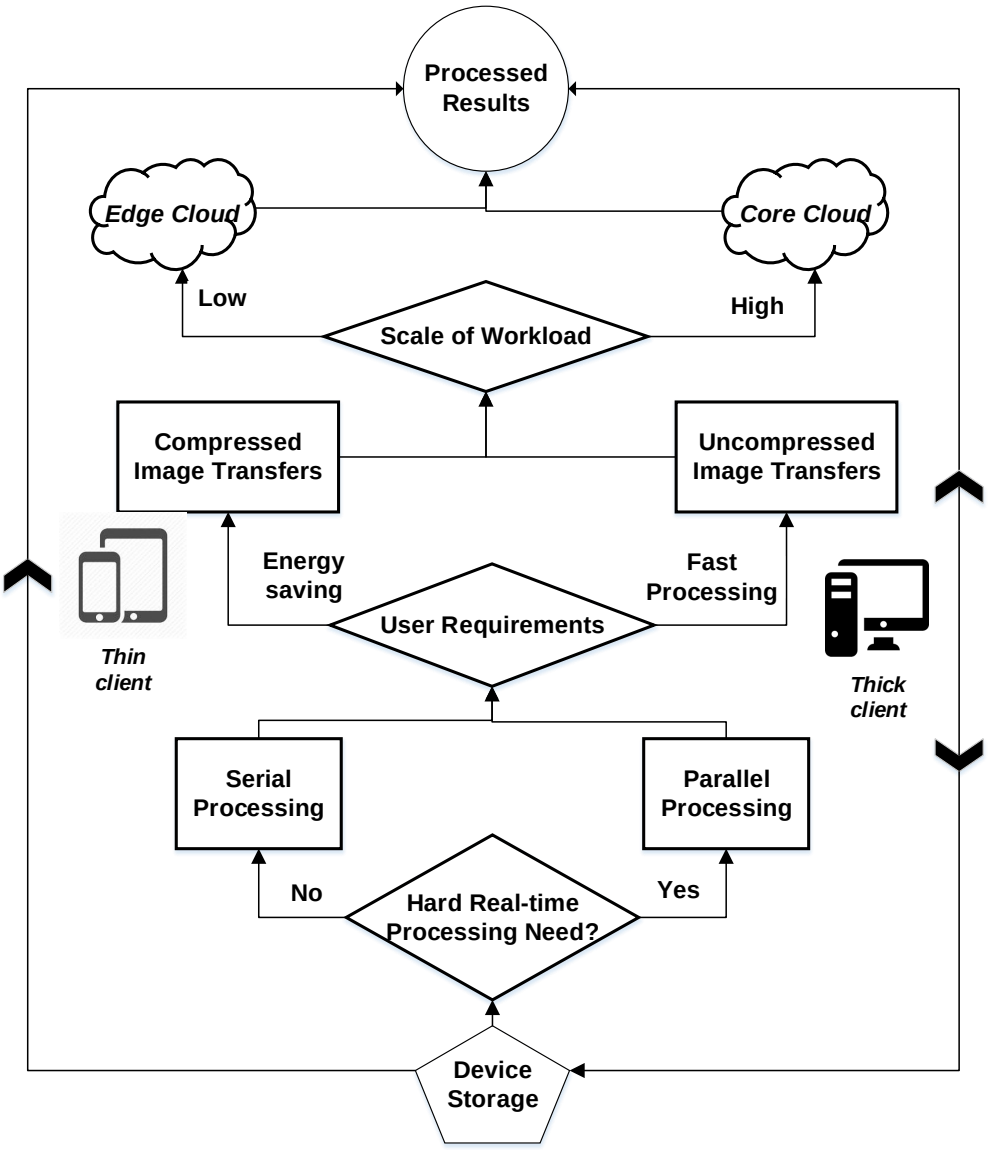


Figure 3.3: Illustration showing the flexible policies-based computation offloading decision-making.

The application logic of the image/photo processing is displayed as Figure 3.3. After the photo is captured, there are multiple decisions that will make a difference on the energy consumption and processing latency. For instance, the transformation of the photos can be performed in parallel or in a sequential/serial manner, depending on real-time processing needs of the users. In addition, the photos can be compressed before being uploaded to the cloud platform. Obviously, the photo size will be smaller and thus it takes less bandwidth to upload but at the expense of additional energy and time

consumption. Without comparison and analysis, it is challenging to decide whether the overall effect is positive or negative. The same problems arise when the results are sent back to the device based on user requirement in the thick client case. Our experiments seek to evaluate the tradeoffs in these various conditions under low-to-high workload scales. The real-time requirement of the application is another factor, i.e., if the face recognition results are required instantly for post-processing on the client side, the workflow has to be optimized. However, if the face recognition results are not required instantly, the results can be simply shown on the server instead of transferring the results back to the client side. In this way, redundant steps can be eliminated and a better performance can be achieved based on the user requirements as well as the client/server capabilities.

Workload allocation to the edge cloud or core cloud considering energy awareness introduces additional challenges for various scenarios. For example, due to a case where energy conservation and fast computation time are desired, processing has to be completed using a cloud platform. However with the remote processing, the additional energy consumption to transfer images also affects the processing latency. Computation offloading onto the edge cloud in this case could save energy and image transfer time, however the edge cloud might have limited resources to handle large workload scales or facilitate parallel processing. The energy and latency metrics thus should be given different priority (or weight) for different workloads so that a reasonable strategy can be selected in the end-to-end steps of the visual data processing.

Algorithm 1 shows our energy and latency aware steps in computation offload decision-making. The *main()* function gets executed first to check whether the user needs to receive the final results from the server as in the thick client case; or whether

Algorithm 1: Offload decision-making

```
Data: Device info: RAM, processor, memory_capacity
Data: Load info: resolution, size_of_load
Data: User requirements: Download, Battery_Saving
Result: The efficient way to save energy and achieve low-latency processing
function create_Thread ()
    /* Create multiple threads on the mobile device to send load balanced data to different servers for processing */
    send ← device_thread_create(processor, {size_of_load,
    number_of_servers});
end
function Offload_decision (EdgeServer)
    /*Decide when and where to offload computation and transfer data */
    if User_Requirement() then
        compress();
    end
    if Workload.isLow() then
        sendToEdge();
    else
        sendToCore();
    end
    transmit();
end
function main ()
    /* Decide best client configuration */
    if Download_result then
        Thick_client();
    else
        Thin_client();
    end
    create_Thread();
    Offload_decision();
end
```

thin client assumptions are relevant on the user side. Once a decision on either the thin or thick client is made, two operations occur subsequently. Firstly, *create_Thread()* ensures that the mobile device is creating multiple threads to start UDP sessions if we have multiple server instances provisioned in the core cloud as represented by variable *number_of_servers*. Based on the need, the value of *number_of_servers* can be configured for parallel or serial transfer to server. Consequently, serial transfers frequently consume less energy, but result in longer processing times. Secondly, function *Offload_decision()* chooses among 'with' or 'without' image compression options for sending the data to the edge or core cloud depending on the scale of workload and the real-time processing user requirement. If the workload is large which means either the number of images or their resolution is high, the mobile devices will send data directly to the core cloud. Note that image compression will help if the workload is low and the user requires

energy conservation. Moreover, to avoid the overloading the edge cloud, the mobile device could periodically monitor edge/core cloud resources and check for availability before transferring data.

Chapter 4

Energy-aware and Sustained Performance Edge Routing

Having a policy-based decision making scheme allows us to make flexible decisions on "*where*" to offload users' visual data processing. However, the question about "*how*" to offload users' data to deliver desired Quality of Application remains open. For example, in some disaster incident response situations users may prefer to have a low-latency processing of their data over a better energy management on power constrained IoT devices. Thus, there is a need in a flexible routing algorithm that works in most disaster incident scenarios and similarly to our decions making scheme takes into account specified user policies.

In this section, we present our Sustainable Policy-based Intelligence Driven Edge Routing (SPIDER) that builds upon recent advances in the geographic routing area to work in challenging disaster-incident conditions.

4.1 Relation of MEC and Edge Routing

We start first by describing the relation between our offloading decision making scheme and the edge routing within our MEC framework. Figure 4.1 shows the life-cycle of our MEC framework, where at the first step users specify their policies such as thin vs. thick clients, compress vs. non-compressed data, sequential vs. parallel process-

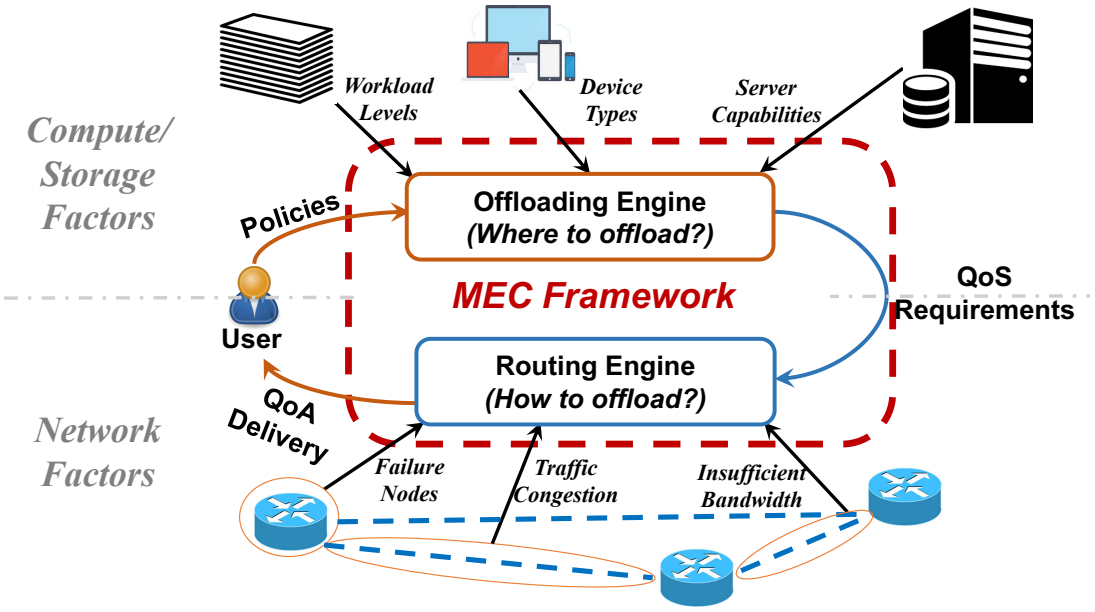


Figure 4.1: Life-cycle of our MEC Framework: add something here...

ing and energy-efficient vs. low-latency computing to our offloading engine. Based on these policies, the offloading engine generates decisions on 'where' to offload users' data that can potentially satisfy their demands. This engine takes into account various compute/storage factors such as different workload levels of physical resources, device types, server capabilities, etc. After that, the offloading engine translates user policies w.r.t. compute/storage factors to network QoS requirements for the edge routing engine. Based on these requirements, the routing engine then controls *'how'* users' data are steered taking into account various network factors such node failures, congestion, lack of bandwidth, etc. Finally, based on delivered QoA levels, users can prefer to try different sets of policies.

In order to satisfy network QoS requirements w.r.t user policies and various network factors such as severe node failures, node mobility, traffic congestion, insufficient bandwidth that are common in MANETs, we outline our SPIDER algorithm in the rest of this section.

4.2 SPIDER Solution Approach

As mentioned earlier, we are interested not only in improving our routing performance sustainability in regions of aftermath disaster but also in making it policy-based to better serve for MEC needs. To this aim, our SPIDER solution approach utilizes the following information:

1. each packet header contains a target region (e.g., destination IP address and its GPS location) and its corresponding forwarding policy (i.e., energy-efficient versus low-latency)

2. each node knows locations and remaining energy levels of all its neighbor, e.g., by periodically beaconing them[1]

3. each node also aware about local obstacles' radius and location detected by the Edge Cloud

Note that SPIDER has no strong assumptions on MANETs' topology such as unit-disk graphs or symmetric links. We remark that our SPIDER improves the baseline performance of the geographic routing and builds upon our previous work on Attractive, Repulsive and Pressure Greedy Forwarding (ARPGF) [34]. Similarly to ARPGF, our SPIDER alternates *Attraction*, *Repulsion* and *Pressure* forwarding modes. When a packet is forwarded in Attraction mode, it attracts to the destination based on its geographic proximity. On the contrary, when the packet is forwarded in Repulsion mode, it can be repealed away from physical obstacles based on its potential function described

[1] In certain cases, beaconing GPS coordinates and neighbors can lead to a poor network energy-efficiency that reduces its lifetime and leads to a worse wireless coverage. To avoid this, one may consider to adjust nodes' beaconing frequencies w.r.t. nodes' mobility to enhance network lifetime and cover larger geographical distances, e.g., 'theater-scale' (about 2 city blocks) or 'regional-scale' ($>$ 30 city blocks) distances.

in Section 4.2.1. Finally, when nodes fail to forward packets in both Attractive and Repulsive modes, packets are forwarded in Pressure mode until either Attractive or Repulsive modes are recovered.

4.2.1 SPIDER Objective

Let consider the following model where node e forwards packet p towards destination d. In this model, node e needs to decide which neighbor should receive p to first of all progress towards d and secondly to balance between neighbor's residual energy and the total latency of p w.r.t. specified policies. Note that the higher latency of p can be due to a longer path as nodes along a shorter path commonly have more drained batteries. We do such balancing by picking e neighbor n with the minimum value of the following objective function:

$$f(n) = \lambda\|\varphi(n)\| + (1-\lambda)\|E(n)\|, \tag{4.1}$$

where $\varphi(n)$ is the potential function of node n that allows us to have theoretical guarantees on packets delivery with $O(3.291)$ approximation of the shortest path [34]; $E(n)$ is a residual energy at node n; and $\lambda \in [0, 1]$ is a parameter to balance between the shortest path approximation φ (to have lower p latency) and its residual energy E level (to get higher overall network energy-efficiency) based on specified MEC policies[2]. In order to compute $\varphi(n)$, SPIDER needs an additional geographic information about physical obstacles such as man-made buildings or natural ponds/lakes and other obstacles that can potentially cause packet drops due to lack of wireless coverage near their

[2] Note however, that minimization of the objective function in Equation 4.1 does not guarantee convergence to the global optimal solution either in terms of packets' latencies or the overall network energy-efficiency. This is due to the fact that our routing solution is a greedy optimization algorithm, i.e., it greedy forwards packets towards the destination. On the contrary, the global optimization needs the full network topology knowledge which is intractable to get in practice due to MANETs' dynamics caused by severe node failures, high node mobility and other disaster-incident challenges.

geographical locations [34]. We discuss how nodes can get such additional obstacles' geo-information of their radius and center coordinates in the next section. Once node e is aware about its local obstacle j radius R_j and center coordinates C_j, it computes φ as following:

$$\varphi(e) = -\frac{1}{dist(e,d)} + \sum_{j=1}^{M} \frac{o_j}{dist(e,C_j)^{\delta}} \tag{4.2}$$

where $dist(e,d)$ is a geographical (e.g., Euclidean) distance between nodes e and d locations; $dist(e,d)$ is a geographical distance between node e and obstacle j center C_j; δ is the attenuation order of obstacles' potential fields that has been shown empirically to give best performance when $\delta \in [1,2]$ [34]; and o_j corresponds to the obstacle j potential intensity induced by the destination node d as following:

$$o_j = \frac{R_j^{\delta+1}}{\delta \cdot (dist(C_j,d) + R_j)^2} \tag{4.3}$$

4.2.2 SPIDER Algorithm

Algorithm 2 outlines how each node forwards packets using either Attractive, Repulsive, or Pressure Greedy Forwarding modes. Node e starts by checking if it has the destination d neighbor (line 43). If not, it then checks if it has any local obstacles known, i.e., $e.\vec{C}$ and $e.\vec{R}$ are not empty (line 47). If so, it proceeds in *Repulsive_Forwarding()* mode (lines 1-10), otherwise it proceeds in *Attractive_Forwarding()* mode (lines 12-21). The *Attraction* mode differs from the *Repulsive* mode mainly by computing $\varphi(e)$ without second term (see Equation 4.2). Finally, if neither attractive nor repulsive forwarding modes are able to find next hop node, e enters the *Pressure_Forwarding()* mode (lines 23-29). This mode is initially proposed by [35] and has been shown to guarantee (both practically and theoretically) packets delivery when the minimization function f

is convex and reaches its minimal value at d. The key idea behind this mode is to forward packets to the closest to the destination neighbor among the least visited neighbors (line 25). As a result, at some point we should be able to recover either *Attractive* or *Repulsive* modes by hitting a node n with $f(n) < f(e)$, where e is a node that enters the *Pressure* mode.

The key difference of our SPIDER algorithm in comparison with ARPGF algorithm is that SPIDER forwards packets based on minimization of the policy-based objective function $f(e)$ (see Equation 4.1). However, such flexibility has a downside as in this case the convexity of f is not longer guaranteed, and hence, packets may not reach the destination d. This is due to the fact that chosen, e.g., best neighbor in terms of its residual energy E can fully disregard shortest path approximation guarantees of φ which has a convexity property required by the *Pressure* mode. This in turn results in violation of the gradient descent to improve φ. To prevent this, we introduce an extra step to choose a *feasible* set of neighbors that guarantees improving of φ (lines 30-39), and hence, the convexity of f to deliver packets.

In the next section, we describe our MEC framework architecture that combines both our offloading decision making scheme and our SPIDER solution approach and features the facial recognition application to improve a visual situational awareness and deep learning to improve SPIDER performance.

Algorithm 2: SPIDER

function $Repulsive\_Forwarding\ (e,\ d)$
  //Get next hop according to φ function
  $nextNodes \leftarrow Feasible\_Neighbors(e, d)$;
  $f_e \leftarrow f(e, P, e.\vec{C}, e.\vec{R})$;
  $nextAddr \leftarrow \arg\min_{n \in nextNodes} f(n, P, e.\vec{C}, e.\vec{R})$;
  if $f_e < f(n, P, e.\vec{C}, e.\vec{R})$ **then**
    return $nextAddr$;
  else
    return NIL;
  end
end function $Attraction\_Forwarding(e, d)$
  //Get next hop according to φ function with only first term
  $nextNodes \leftarrow Feasible\_Neighbors(e, d)$;
  $f_e \leftarrow f(e, P)$;
  $nextAddr \leftarrow \arg\min_{n \in nextNodes} f(n, P)$;
  if $f_e < f(n, P, e.\vec{C}, e.\vec{R})$ **then**
    return $nextAddr$;
  else
    return NIL;
  end
end function $Pressure\_Forwarding(e, d)$
  //Always return a best effort next hop for forwarding
  $visits_{min} \leftarrow \min_{n \in Nbrs(e)} P\_visits(n)$;
  $Candidates \leftarrow \{nextAddr \in Nbrs(e)$ and $P\_visits(n) == visits_{min}\}$;
  $P\_visits(n) \leftarrow P\_visits(n) + 1$
  $nextAddr \leftarrow \arg\min_{n \in Nbrs(e)} f(n, P, e.\vec{C}, e.\vec{R})$;
  return $nextAddr$;
end function $Feasible\_Neighbors(e, d)$
  //Get set of next hop according toward destination
  $\varphi_e \leftarrow \varphi(e, P)$;
  foreach $n \in Nbrs(e)$ **do**
    $\varphi_n \leftarrow f(n, P, e.\vec{C}, e.\vec{R})$;
    if $\varphi_n < \varphi_e$ **then**
      $nextNodes \leftarrow nextNodes \cup n$;
    end
  end
  return $nextNodes$;
end function *main ()*
  //Upon receiving a packet P at node e, algorithm decides to which neighbor of e send P next
  if $d \in Nbrs(e)$ **then**
    $nextAddr \leftarrow d$;
  else
    $nextAddr \leftarrow NIL$;
    if $e.\vec{C} \notin \emptyset$ `and` $e.\vec{R} \notin \emptyset$ **then**
      $nextAddr \leftarrow Repulsive\_Forwarding(e, dst)$;
    if $nextAddr! = NIL$ **then**
      $nextAddr \leftarrow Attraction\_Forwarding(e, dst)$;
      if $nextAddr == NIL$ **then**
        $nextAddr \leftarrow Pressure\_Forwarding(e, dst)$;
    $send(P, nextAddr)$;
  end
end

Chapter 5

MEC Framework Architecture

In this chapter, we describe our MEC Framework architecture shown in Figure 5.1 that is comprised from three logical components: the MEC Framework itself; its offloading engine that interacts with user IoT and the dashboard; and the MEC routing engine that uses deep learning service available in the Core Cloud to coordinate our SPIDER routing in MANET.

5.1 MEC Framework at the Edge.

The first core logical component of our architecture is our MEC Framework, that we place at the Edge Cloud for a low-latency interactions with user IoT devices through the gateway. Our MEC Framework has two main service components - offloading and routing engines that synergistically decide 'where' and 'how' to offload IoT data processing within the hybrid Core/Edge Cloud, respectively.

Particularly, our offloading engine is needed to augment IoT with demanded storage and compute resources while taking into account specified user policies and various compute/storage factors such as workload levels, device types, server capabilities, etc. Our routing engine in turn is needed to steer traffic for achieving desired network QoS based on specified user policies and diverse network factors such as failure nodes, traffic congestion, insufficient bandwidth, etc.

5.2 MEC Offloading Engine, User Dashboard and IoT.

sec The IoT such as security cameras, civilian smart phones, and aerial perspectives collect patients' visual data. This data can be later transferred to the hybrid Edge/Core Cloud using an offloading engine. When the facial recognition application is opened by the user, offloading engine starts by providing the decision making interface. After collecting users' needs, the offloading engine make the user policies. Then the IoT starts processing to offload which uses the instructs (i.e., compressed/ not compressed, thin/ thick client, parallel/ sequential processing, and edge/ core cloud) from the offloading engine. To this end, IoT devices offload the application to the user dashboard at the Edge server, to provide low-latency and energy-efficient processing. The Core Cloud database stores patients' visual data, which shares caching with local Edge database. The stored visual data is used for matching & verification step in facial recognition application (see Fig. 3.1).

5.3 MEC Routing Engine, SPIDER and Deep Learning.

Upon receiving network QoS guidance as well as user policies from the offloading engine, our routing engine instructs corresponding user IoT devices on how to send their data. To this end, the routing engine sends to these IoT devices specific λ parameters that need to be stored in their data packet headers for the later use within our SPIDER objective function f (see Equation 4.1). Moreover, based on geographic locations of user IoT devices, there is a need to learn geographic environment obstacles such as man-made buildings or natural lakes or ponds to improve the geographic routing performance of our SPIDER in this area. We remark that such obstacles can cause packet

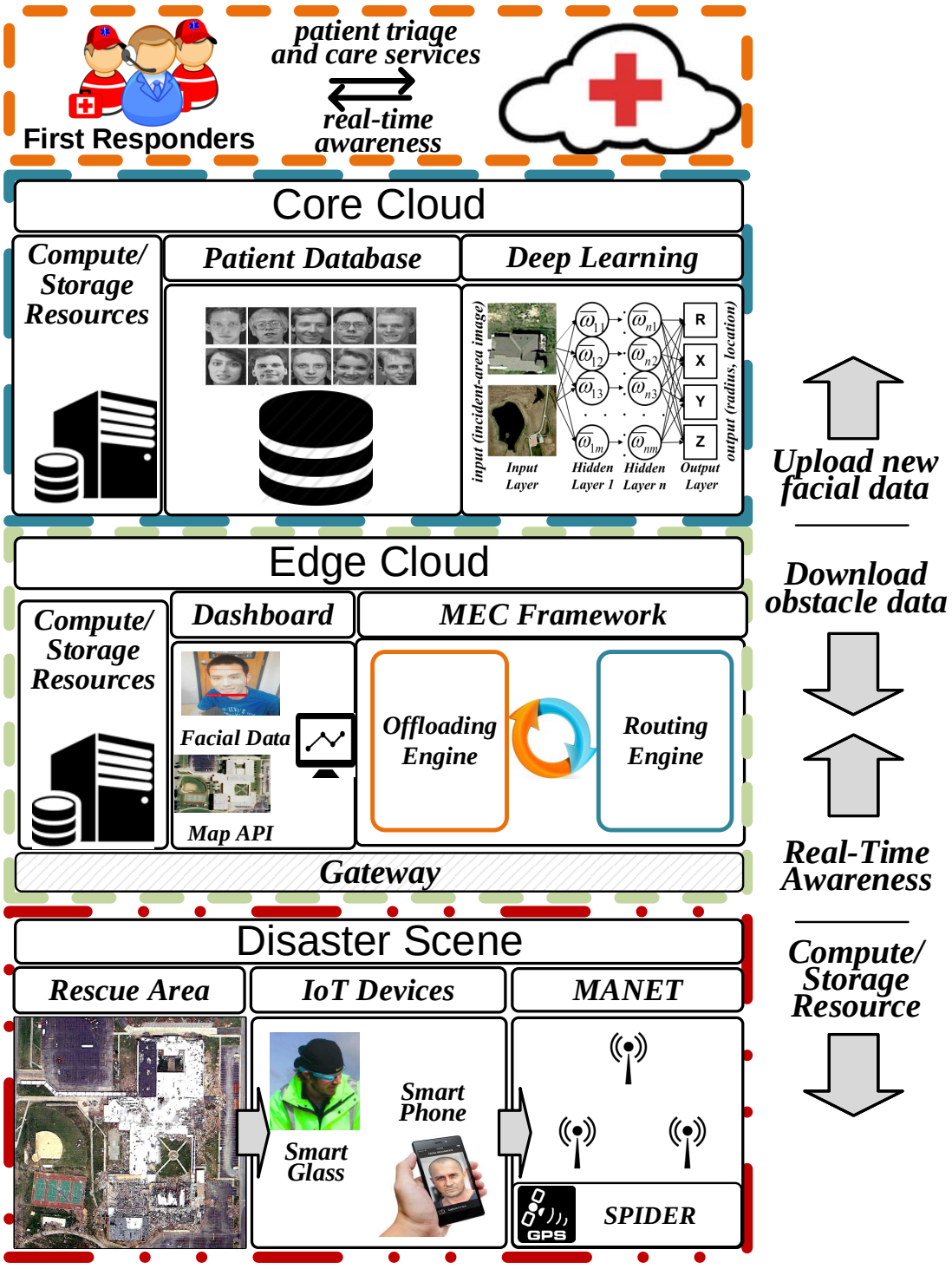


Figure 5.1: Illustration of our MEC Framework architecture that consists of three main logical components: the MEC Framework itself; its offloading engine that interacts with user IoT and dashboard; and the MEC routing engine that coordinates our SPIDER routing in MANET powered by deep learning in the Core Cloud.

drops due to lack of wireless connectivity in their proximity. After that, our routing engine propagates information about these obstacles through the gateway to MANET

in the disaster-incident scene. Node n in MANET stores information about only those obstacles that locate in two obstacle's radius proximity from n to compute $\varphi(n)$ (see Equation 4.2) later [34].

To learn information about gepgraphic obstacles, our routing engine uses the Core Cloud deep learning detectors and a publicaly available map API that contains satellite imagery of the disaster-incident scene. As map API is also available at the user dashboard, our routing engine may detect obstacles even without using the Core Cloud (i.e., offline). However, the most accurate deep learning detectors [32, 33] today may be intractable to run on the constrained Edge Cloud servers. Thus, we recommend to avoid deep learning at the Edge Cloud servers and use it within the Core Cloud instead. To this aim, collected and partly labeled training samples need to be uploaded to the Core Cloud for supervised or semi-supervised deep learning [39] to enhance performance of detectors in future.

Chapter 6

Performance Evaluation

In this chapter, we evaluate our energy-aware and low-latency framework using variety policies such as low-to-high workloads, thin or thick clients, and edge cloud or core cloud. Then using NS-3 (Network Simulator 3), we simulate SPIDER routing in the Joplin area under low-to-high failure and low-to-high mobility scenarios.

6.1 Visual Data Processing Evaluation

To evaluate the impact of flexible data handling such as e.g., handling user preferences for real-time versus energy efficient processing using MEC framework featuring the facial recognition application and our decision-making algorithm with experiments in a realistic edge and core cloud testbed.

6.1.1 Experimental Settings

Figure 6.1 shows our experimental testbed setup where we use a local U. of Missouri server resource [14] for an edge cloud, and we use 10 GENI server instances at New York University (NYU) campus for the core cloud. Our edge cloud server has 70GB of RAM, 12 cores with a bandwidth of approximately 90 Mbps. Each of the core cloud servers have 1 core and 1 GB of RAM, and we connect to them at a bandwidth of approximately 900 Mbps. We use ASUS Zenpad tablet with 2 GB RAM, 1.33 GHz Atom Z3735 processor and 8 hours of battery life (when under common use) as our

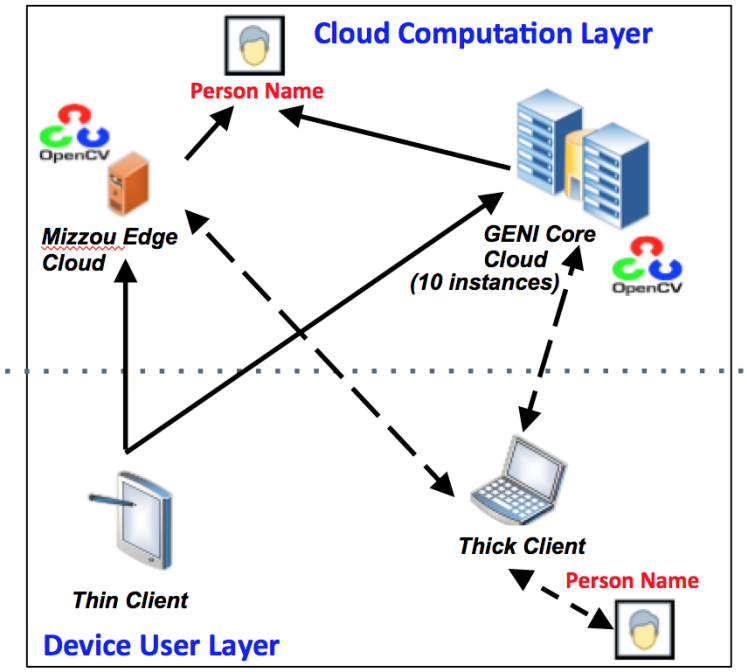


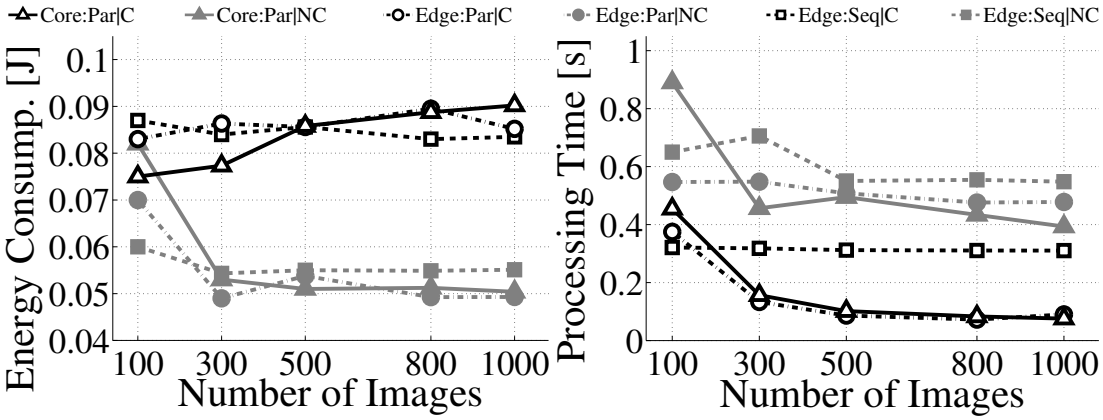
Figure 6.1: Testbed edge cloud is a server on Mizzou campus and our core cloud includes 10 GENI server instances at New York University (NYU) campus.

mobile device that runs the facial recognition application described in Section 3.

6.1.2 Comparison methods and metrics.

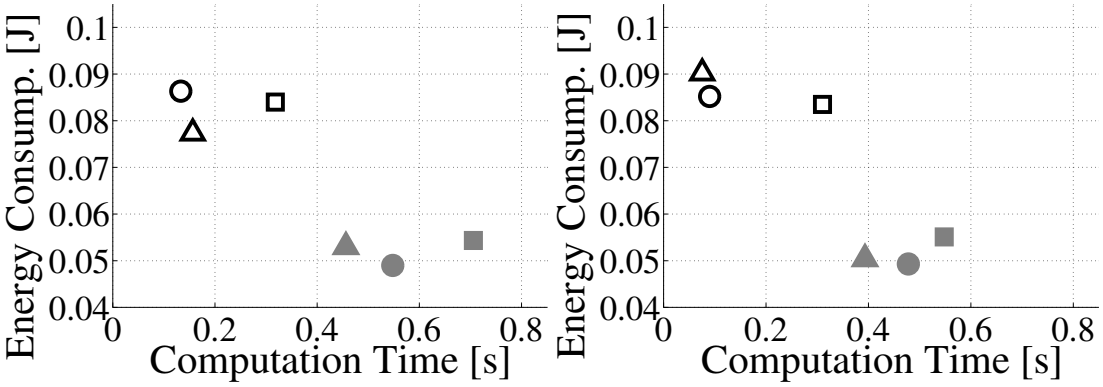
We compare cases where *thin* client or *thick* client configurations are used. Particularly, our thin client configuration at a user assumes all of the processed images are stored and viewed at a remote cloud resource, whereas a thick client configuration assumes processed images are downloaded and further post-processed at the mobile device level. We start our experiments by offloading application threads on the mobile device to the cloud computation layer for remote processing as shown in Figure 6.1. We vary the processing workload from 100 to 1000 images of 2048 x 1536 pixels size that are transferred to the remote server side using the UDP protocol. We also use different MEC policies including parallel (Par) processing versus serial or sequential (Seq) and data compression (C) versus no compression (NC) policy. We use the Android-based Pow-

erTutor utility [15] to profile and estimate *energy consumption* of our facial recognition application (Metric: Joules) within the testbed setup. Our end-to-end *processing time* includes the time needed for an image export/import to edge or core cloud and its remote processing (Metric: Processing Time in Seconds). The results of *energy consumption* and *processing time* are per image in the graphs. On the IoT device, we are using thin (see *top*) and thick (see *bottom*) clients when offloading to the Core or Edge clouds with parallel (Par) or sequential (Seq) processing policies as well as with data compression (C) or no data compression (NC) policies.



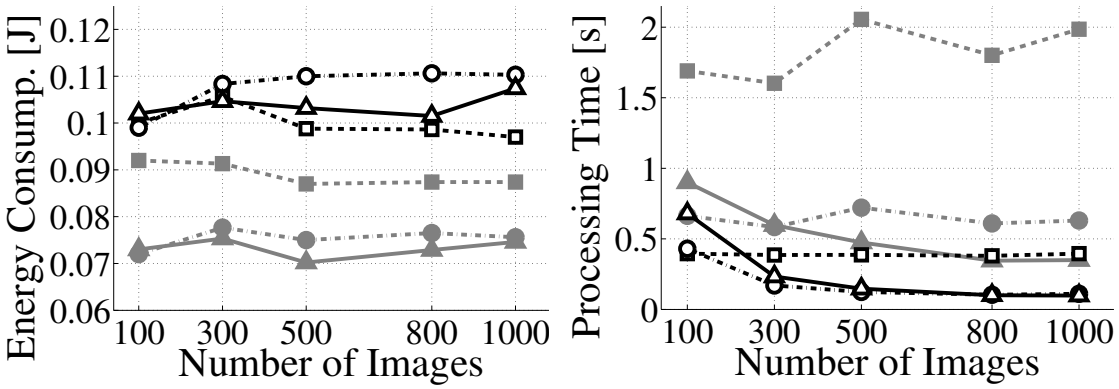
(a) Thin Client - Energy

(b) Thin Client - Latency



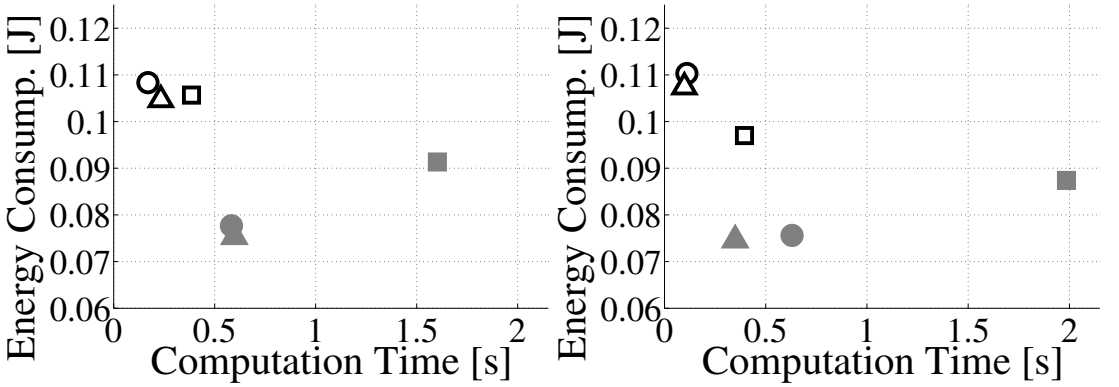
(c) Thin Client - Low Workload

(d) Thin Client - High Workload



(e) Thick Client - Energy

(f) Thick Client - Latency



(g) Thick Client - Low Workload

(h) Thick Client - High Workload

Figure 6.2: Average energy consumption (a,e) and processing time (b,f) per image with their trade-offs under a low workload of 300 images (c,g) and under a high workload of 1000 images (d,h)

6.2 Discussion

6.2.1 Policy-based Optimization

We start our MEC framework evaluation by discussing its optimal policy sets (defined by a combination of decision parameters in Figure 3.3) that cover diverse user's demands and input data scale. Based on energy consumption results in Figures $6.2a$ and $6.2e$, we observe the following policies needed for the *maximum operational time of the mobile device*: for both *thin* and *thick* client configurations, we need to use *parallel* processing over *non-compressed* data. This observation is due to the fact that data compression significantly utilizes CPU resources of the mobile device, and the

sequential data offloading further introduces additional energy consumption for data export/import. Note however in this case, offloading either to the edge cloud or core cloud has no impact on the energy consumption within the mobile device. However, processing time results shown in Figures $6.2b$ and $6.2f$ indicate how our optimal MEC policies for the *minimum end-to-end processing time* have changed. Particularly, for both *thin* and *thick* client configurations, we now need to use *parallel* processing over *compressed* data. Moreover, when the workload scale is low (e.g., ≤ 500 images), we can further speed up our remote processing by offloading to the edge cloud versus offloading to the core cloud. This difference is due to the higher latency of transferring data to the cloud, which degrades as the workload scale increases; in this case, the edge cloud needs more time to process all the data than the core cloud. Note also how in both cases, a *sequential* offloading policy is worth simultaneously for both the energy consumption and the processing time benefits. However, this policy is needed for live image data (e.g., video streams), where new frames are sequentially captured. Moreover, for both thin and thick client configurations, improving interactivity require more energy consumption in all cases.

6.2.2 Engineering Trade-offs and Pareto Optimality

In practice, users can also benefit from considering an acceptable performance for the reasonable energy consumption instead of only focusing on a single factor as discussed previously. Below, we show how different policy selections can be a part of the Pareto optimal MEC framework strategy for different application energy consumption and processing time trade-offs. Specifically, when observing Figures $6.2c$ and $6.2g$ of a low workload scale (i.e., when processing $\approx$300 images), we can see how *parallel* pro-

cessing of both compressed and non-compressed data at the edge or core cloud are part of the Pareto optimality for both *thin* and *thick* client configurations. More concretely, using *thin* client as an example, *parallel* processing of both compressed and non-compressed data at the core cloud could be among the top two optimal solutions. Overall, both of them could achieve relatively low energy consumption and short end-to-end computation time, but each of them has special advantages. Processing with compressed data consumes 65% less computation time compared to processing of non-compressed data, which requires 46% more energy consumption. Meanwhile, processing with non-compressed data may cost 31.5% less energy consumption, but it takes 191.4% more computation time for end-to-end process. In certain situations, users could choose the optimal solution based on their specific processing demands. To sum up, all of these policy combinations do not result in application performance cases that simultaneously degrade in both the energy consumption and the end-to-end processing time aspects.

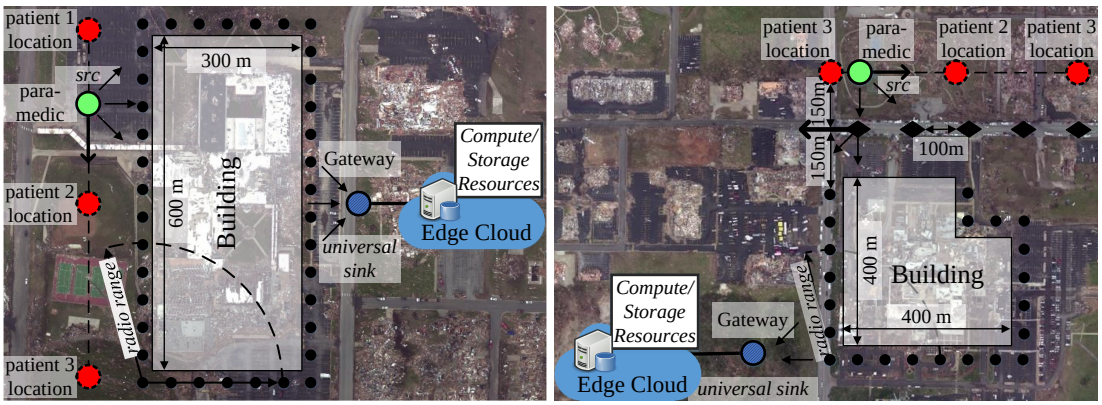
The same however does not hold for a high data workload (i.e., when processing $\approx$1000 images). Particularly, observing Figures $6.2d$ and $6.2h$, we noticed how *parallel* processing of both compressed and non-compressed data at the edge cloud is not part of the Pareto optimal solution set when a *thick* client configuration is used. In this scenario, using *parallel* processing of non-compressed data at core cloud is the optimal solution since it has the lowest energy consumption and third shortest end-to-end computation time (only consumes 83% more computation time compared with the shortest solution but saves 49% energy consumption). The reason for this result is because of the fact that the edge processing time dominates higher data export/import latencies to the cloud. Thus, computation offloading to the edge cloud under a high data workload for

thick clients is always suboptimal to the core cloud offloading for our facial recognition application context.

6.3 Edge Routing Evaluation

To evaluate our SPIDER performance, we use realistic disaster-incident scenarios with severe node failures and high node mobility in NS-3. We then compare its performance with the flexible stateless greedy routing GEAR protocol [28]. WE also compare our SPIDER with common stateful ad-hoc routing solutions: the known reactive Ad-Hoc On Demand Distance Vector (AODV) protocol [40]; and the Hybrid Wireless Mesh Network (HWMP) protocol 802.11s standard [41] that combines reactive (by using AODV) as well as proactive routing (by using spanning trees).

6.3.1 Simulation Settings.



(a) (b)

Figure 6.3: First scenario (a), we evaluate our approach under severe failures, and in the second scenario (b), we evaluate our approach under high mobility.

For our comparison, we implement on the ns-3 simulator [42]. We use realistic disaster scenes of damaged by tornado Joplin High School and Joplin Hospital buildings in Joplin, MO (2011) Figure 6.3 (a) and at the Joplin Hospital (b) buildings in Joplin, MO (2011) to evaluate performance of stateless greedy forwarding algorithms under

mobility and (severe) node failures. To recreate these disaster scenes, we used the available satellite maps of Joplin, MO, tornado response imagery [43]. We assume that the information regarding a damaged buildings ($e.g.$, its center coordinates and radius) was provided from the edge-cloud through a Gateway using proposed obstacle detector (see Section 5) on pre-uploaded satellite maps.

In our disaster-incident scenario, a paramedic acts as a source sending data to the gateway (universal sink) over a resilient ad-hoc network. Video streams gathered on-site are sent over a UDP session to the dashboard located in an edge-cloud for further data processing in conjunction with a core cloud. We simulate the 5 Mbps high-definition images transmission over a UDP connection from a heads-up display device worn by a paramedic $e.g.$, Google Glass acting as a visual data source. The paramedic stays for 3 minutes at each patient location and moves at a jogging speed (2.8 m/s) between these locations. The simulation is designed to cause a geographical routing to face a local minimum when the paramedic source is near the second or third patient locations. Aside from the source mobility, in the node failure simulation scenario (see Figure 6.3a), nodes around an obstacle can fail for the next 30 seconds[1] with a probability sampled from the interval $[5\%, 50\%]$ (from low to severe node failures). Under these failure conditions, the goodput degrades due to losses ($e.g.$, caused by packet collisions) that increase with the path length or path reconstruction of the stateful routing approaches. Note that when nodes fail for continues periods, any "store and forward" solution is inadequate [44, 45]. We then evaluate impact of node mobility in the second simulation scenario (see Figure 6.3b), where paramedics can communicate with the gateway only through moving on the road vehicles with the speed sampled from the interval $[5, 20]$

[1]Such behavior is expected due to possibility of an intermittently available power supply, or due to a physical damage caused by rescue workers near the disaster scene.

m/s (from low $\approx$ 10 mph to high $\approx$ 40 mph mobility cases). Under these mobility conditions, any stateful routing solutions ($i.e.$, which relies on the network topology knowledge such as spanning trees) has poor performance [46]. We use a simple energy model in which every node starts with the same initial energy budget, and transmitting or receiveing a packet consumes one unit of energy. We do not count energy consumes by control packets or beaconing hello message.

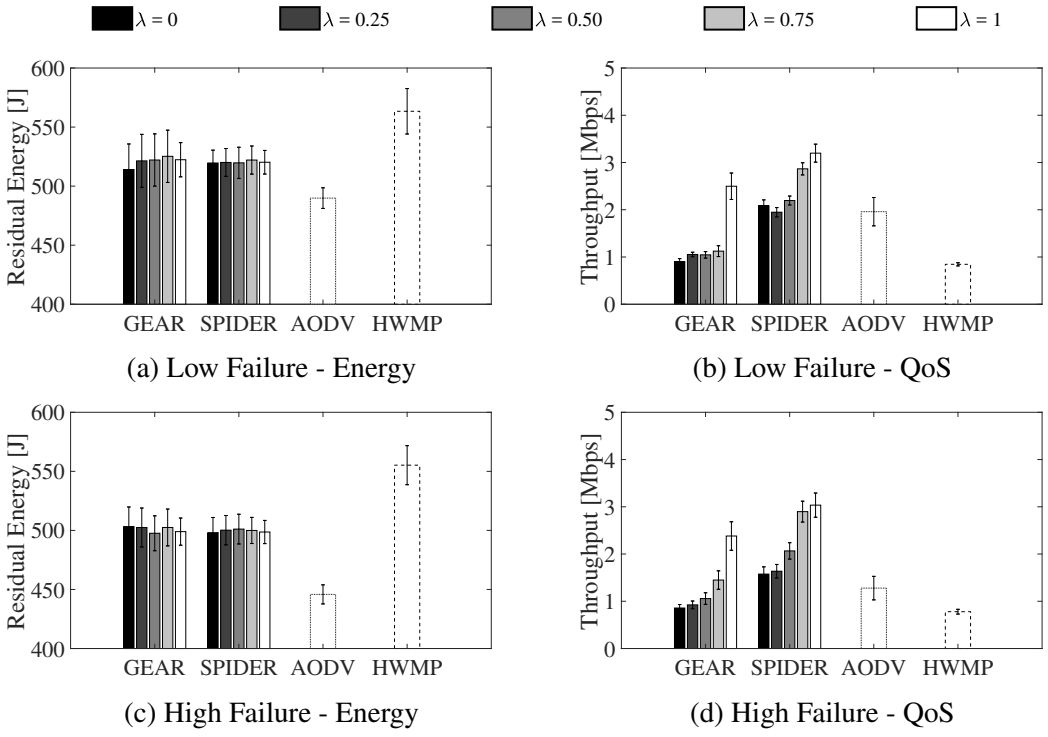
Finally, nodes are placed on a grid ranging from 50 - 150 m step, each node has a radio range of 250 m, and an obstacle (a building) is located approximately in the center of this grid. Each node has roughly $3 - 10$ neighbors for resilience purposes. Table 6.1 summarizes our simulation details.

Table 6.1: Simulation Environment SettingEs

Topology:		**Physical/Link layers:**	
Number of nodes:	30 - 40	Frequency:	2.4 GHz
Grid placement:	50 - 150 m	Tx power:	20 dBm
1^{st} obstacle size:	600 x 300 m	Tx gain:	6 dB
2^{nd} obstacle size	400 x 400 m	Rx gain:	0 dB
Radio range:	250 m	Detection threshold:	-68.8 dBm
Avg node degree:	$\approx 3 - 10$	Delay prop. model:	CONSTANT SPEED
Overall settings:		Loss prop. model:	TWO-RAY
Node failure period:	$\approx$ 0.033 Hz	Technology:	802.11g/s
Node failure probability:	0.05 - 0.5	Modulation:	OFDM
Mobile nodes speed:	5 - 20 m/s	Data rate:	54 Mbps
Time at each location:	180 sec	**Transport/App layers:**	
Src speed:	2.8 m/s	Transport protocol:	UDP
Simulation time:	720 - 780 s	Payload:	1448 bytes
Beaconing frequency:	1 - 4 Hz	Application bit rate:	5 Mbps

6.3.2 Comparison methods and metrics.

To compare with stateful ad-hoc routing solutions, we use the known reactive Ad-Hoc On Demand Distance Vector (AODV) protocol [40]. We also use Hybrid Wireless Mesh Network (HWMP) protocol of 802.11s standard [41] which combines reactive (with AODV) as well as proactive routing (using the spanning tree algorithm). We also use different routing policies including $\lambda = 1$ to obtain best sustainable results, $\lambda = 0$ to obtain best energy conservation, and $\lambda = 0.25$, 0.50, and 0.75 to obtain balanced efficient routing. We use energy average of all nodes in network and estimate *network residual lifetime* (Metric: Joules) of our routing approach. Our end-to-end *throughput* includes the time needed and packets deliver for images transmit to Gateway (Metric: Mbps).

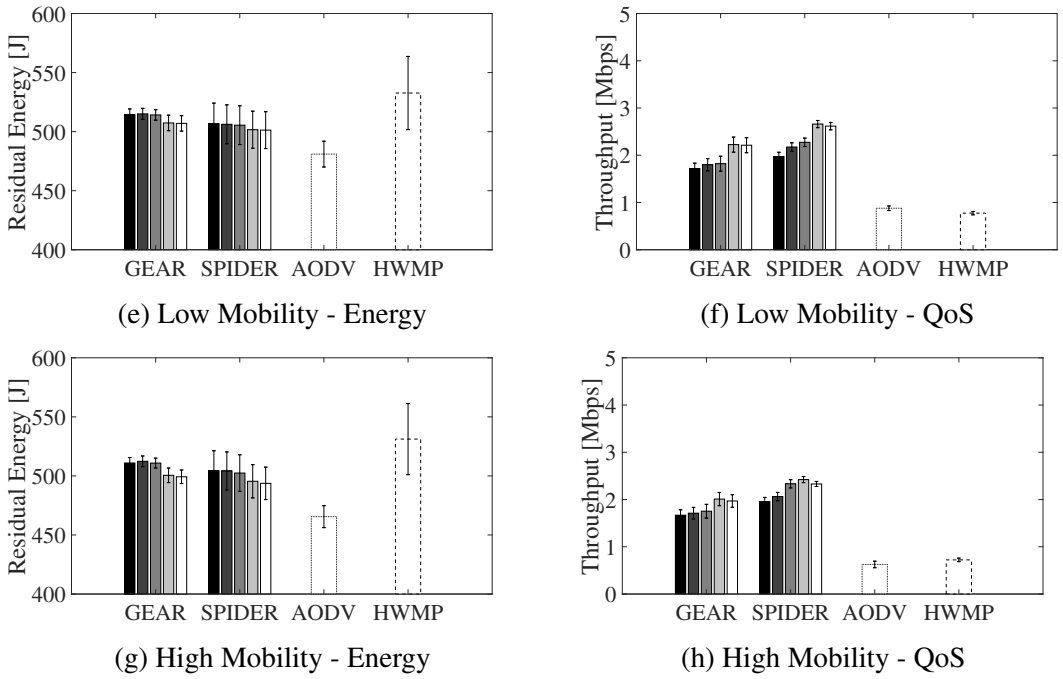


(a) Low Failure - Energy

(b) Low Failure - QoS

(c) High Failure - Energy

(d) High Failure - QoS



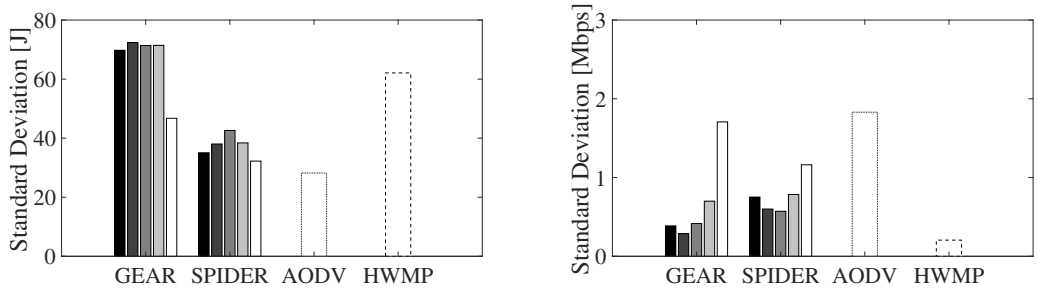
(e) Low Mobility - Energy

(f) Low Mobility - QoS

(g) High Mobility - Energy

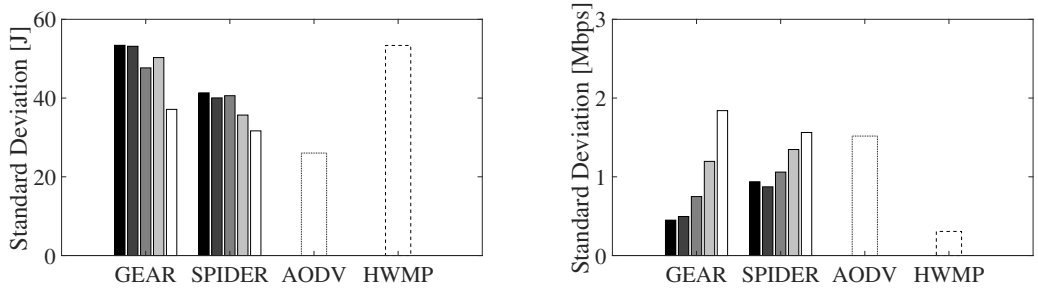
(h) High Mobility - QoS

Figure 6.4: Average residual energy (a,e,c,g) and data delivered (b,f,d,h) per cases with their trade-offs under variety λ



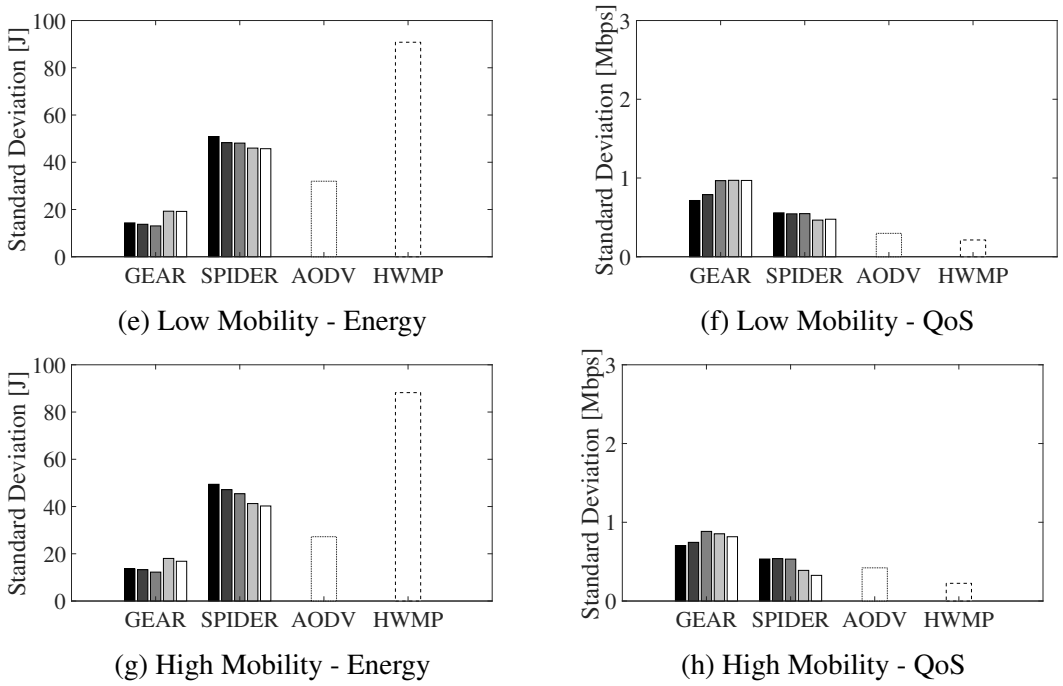
(a) Low Failure - Energy

(b) Low Failure - QoS



(c) High Failure - Energy

(d) High Failure - QoS



(e) Low Mobility - Energy

(f) Low Mobility - QoS

(g) High Mobility - Energy

(h) High Mobility - QoS

Figure 6.5: Standard deviation residual energy (a,e,c,g) and data delivered per second (b,f,d,h) per cases with their trade-offs under variety λ

6.4 Discussion

6.4.1 Sustainable Policy-based

We start our SPIDER evaluation by discussing its optimal policy sets (defined by a combination of decision parameters in Equation 4.1) that cover variety policy balance between sustainable and energy consumption. Based on energy remaining results for several cases of λ in Figures $6.4a$ and $6.4e$, we observe the following policies needed for the *maximum lifetime of the network*: for our application, we can use λ = *0 or 1* over *Low Failure and Low Mobility* scenarios. However, in the case of *High Failure and High Mobility*, we need to use λ = *0 or* λ = *0.25* (Figures $6.4c$ and $6.4g$. This observation is due to the fact that too much emphasis on energy on next single hop force routing to choose longer path. Hence the overall energy efficiency get worse. The same however does not hold for Low Failure case (i.e., when nodes fail 5% of all the

time). Particularly, observing Figures $6.5a$ and $6.5b$, we noticed how $\lambda = 0 or 0.25$ is not part of the optimal solution set. In this scenario, using *SPIDER with* $\lambda = 0.5$ routing is the optimal solution since it has the highest deviation residual energy and third shortest deviation throughput. Specifically, when observing Figures $6.4b$ and $6.4d$ of Failure cases (i.e., when nodes fail 5% and 50% of the time they receive a packet to forward) and Figures $6.4f$ and $6.4h$ of Mobility cases (i.e., when node mobility 5 m/s and 20 m/s), we can see how different λ affect the throughput overall. $\lambda = 1$ is optimal for both *Low Failure* and *High Failure* scenarios. More concretely, using *Failure* cases as an example, $\lambda = 0.25$ *or* $\lambda = 0.75$ of SPIDER routing at the edge cloud could be among the top two optimal solutions. Overall, both of them could achieve relatively high network lifetime and high throughput (e.g. short end-to-end latency), but each of them has special advantages. Thus, in general, we achieve energy efficiency when $\lambda = 0.25$ and sustainable QoS when $\lambda = 1$ for offloading process in our facial recognition application context. To sum up, it is not necessary to choose a specific option for routing, but changing of λ value in real-time situation helps to adapt scenario.

6.4.2 Engineering Trade-offs

Below, we show how our proposed SPIDER is better for visual application in MEC framework strategy for flexible consideration in energy consumption and QoS trade-offs.

For residual results for several disaster scenarios in Figures $6.4a$, $6.4c$, $6.4e$, and $6.4g$, we observe that GEAR is more energy efficiency in all cases of λ compare to SPIDER. However, the standard deviation in Figures $6.5a$, $6.5c$, $6.5e$, and $6.5g$ show higher range of results for GEAR. In practice, it's not enough to choose only energy efficiency

for affective routing but also considering throughput of routing. For low node failures (5%), SPIDER delivers all packets with a UDP throughput of ≈ 3 Mbps (Figure 6.4b). GEAR instead has a 20% failure rate in delivering packets. When GEAR enters the recovering mode it uses planarization which, in turn, can significantly stretch paths. As a result, GEAR shows lower UDP throughput (< 2 Mbps) than SPIDER, and only 20% of the time it shows similar performance. Under severe node failure conditions (nodes fail 50% of the time they receive a packet to forward), we observe similar behaviors (Figure 6.4d): GEAR experiences lower UDP throughput 40% of the time compared to SPIDER. Under such severe failures, both GEAR and SPIDER fail to deliver packets $\approx$ 45% and 35% of the time. For low node mobility (5 m/s), both GEAR and SPIDER deliver all packets with a UDP throughput of $\approx 2 - 3$ Mbps (Figure 6.4f). However, when GEAR enters the recovering mode near patient location 3 (after ≈ 500 sec), due to its planarization (which stretches paths), it shows lower UDP throughput (≤ 2 Mbps) than SPIDER. At the same time, 60% of the time (first 500 sec) it shows similar performance. Under high node mobility conditions (20 m/s), we again observe similar behaviors (Figure 6.4h): GEAR experiences lower UDP throughput 60% of the time compared to SPIDER, caused again by the planarization when GEAR faces a local minimum. Under such high mobility, both GEAR and SPIDER are able to still deliver all packets, which makes geographical routing more attractive disaster-incident response activities which benefit from the real-time situational awareness. Even though both AODV and HWMP have advantages over pure proactive stateful routing solutions, in a challenged disaster scenario they do not show acceptable throughput level, leading to service outages caused by disconnections (from 20% to 90% percent of the time). Recent solutions in stateful greedy forwarding literature can help cope with some disaster

incident challenges [47, 48]. For example, recent stateful greedy forwarding solutions have shown promising results under severe node failures [48]. However, we found no stateful greedy forwarding algorithm which can cope with both severe node failures and high mobility. The superior performance of SPIDER arise due to its knowledge about a static physical obstacle located within the disaster scene, which in most cases allows local minima avoidance by using our proposed *Repulsion* forwarding. Hence, GEAR is better in term of energy because it intent to push less data (due to the affect of local minima and stretch the path) but our solution is better in term of flexibility.

Chapter 7

Future work

The area of research that deals with performance and energy-awareness involving Mobile Edge Computing is fairly new. Moreover, enabling flexible policy-based sets are important to incident supporting. Therefore, this thesis has addressed only a small subset of the numerous challenges involved in provisioning and analysis of energy-awareness and performance trade-off purposes in automated frameworks.

7.1 Virtual Network Embedding

A concern we did not consider in Chapter 3 is the scale-up challenge of project. Therefore, we want to apply Virtual Network Embedding (VNE) to allocate virtual resource on physical infrastructure. This allows the facial recognition application processes mapped on best-suit nodes in network. For example, a virtual node can, in principle, be hosted by any available node to performance a process in application when the offloading is on the way to the edge or core cloud. Moreover, if there are changing in network status, VNE should allow to applicable in larger scale and relocate to mapped Virtual Network Resource (VNR) in online manner.

7.2 Smart Controller

Other concern we did not consider in Chapter 4 is the "Smart Controller" of variety policy-based sets. For future work, we plan to investigate Artificial Intelligence al-

gorithms to make computation offloading decisions and routing decisions considering multiple edge cloud resources. This also helps satisfy user requirements in visual data processing involving more constraints such as user mobility, edge available resource and processing time. As part of future work, we are planning on development an efficient Deep Learning algorithm, which will operate on publicly available or pre-downloaded maps API ($e.g.$, Google Maps) to compute information about known static obstacles (i.e., their center coordinates and radius) within disaster-incident area. With such extensions to our work, AI involved in controller gains efficient and adaptive policy issues can be overcome.

7.3 Practical Protocol

Last but not least, in case of developing practical product, we plan to use goTenna and create peer-to-peer to build a mesh network for incident response. For example, in the article [49] claims "Puerto Rico lost 93% of its connectivity after Hurricane Mariasystems will be down for months to come. PR Reconnects is building out a goTenna Mesh network there." Their product tends to create mesh network without infrastructure in large area and able to send simple but efficient message to the destination. On the other hand, their product still have problem of limited power resource. Based on our improvement, our protocol could even send a streaming video or sequential images to the destination with sustainable performance, low-latency and energy-awareness.

Chapter 8

Conclusion

In this thesis, we studied how the mobile edge computing (MEC) paradigm can provide flexibility to users who desire energy conservation over low-latency or vice versa in visual IoT-based application data processing. Our work was based on the rationale that computing should happen in the proximity of data sources, and cloud services especially moved closer to the network edge can present opportunities to meet user requirements in terms of energy consumption and fast processing times. Using a facial recognition application that we developed for use on mobile devices, we were able to demonstrate cases where thin client or thick client configurations are more effective at low-to-high visual data processing workloads, and how offloading policies could affect the energy efficiency or low latency user requirements. Particularly, we found from the results that the edge cloud offloading policy for thick clients is always sub-optimal in comparison to the core cloud offloading under high workloads. However, it was not the case for thin clients under similar conditions. Also, we addressed the lack of sustainable and flexible routing approaches for offloading facial recognition application in MANETs to trade-off between energy-awareness and low-latency data transferring to an edge cloud gateway within the lost infrastructure area. Specifically, we presented our Sustainable Policy-based Intelligence Driven Edge Routing (SPIDER) that builds upon recent advances in the geographic routing area. To improve its baseline geographic routing performance, SPIDER uses additional geographic knowledge that we obtain from the publicly available satellite imagery of the rescue area and deep

learning detectors in the core cloud. To balance between energy-awareness and low-latency data transferring in a best-effort manner, our SPIDER uses tunable objective function. Considering variety cases of actual incident-supporting scenarios, we have shown how our SPIDER is more flexible and is more sustainable than other stateless *geographic routing* solutions (i.e., GEAR and GPGF) as well as stateful reactive mesh routing (i.e., AODV and HWMP).

Bibliography

[1] Hyunduk Kim, Myoung-Kyu Sohn, Dong-Ju Kim, and Nuri Ryu. User's gaze tracking system and its application using head pose estimation. In *Artificial Intelligence, Modelling and Simulation (AIMS), 2014 2nd International Conference on*, pages 166–171. IEEE, 2014.

[2] John Gillis, Prasad Calyam, Olivia Apperson, and Salman Ahmad. Panacea's cloud: Augmented reality for mass casualty disaster incident triage and coordination. In *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual*, pages 264–265. IEEE, 2016.

[3] Joshua C Klontz and Anil K Jain. A case study on unconstrained facial recognition using the boston marathon bombings suspects. *Michigan State University, Tech. Rep*, 119(120):1, 2013.

[4] Arif Ahmed and Ejaz Ahmed. A survey on mobile edge computing. In *Intelligent Systems and Control (ISCO), 2016 10th International Conference on*, pages 1–8. IEEE, 2016.

[5] Yaser Jararweh, Ahmad Doulat, Omar AlQudah, Ejaz Ahmed, Mahmoud Al-Ayyoub, and Elhadj Benkhelifa. The future of mobile cloud computing: Integrating cloudlets and mobile edge computing. In *Telecommunications (ICT), 2016 23rd International Conference on*, pages 1–5. IEEE, 2016.

[6] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. Towards wearable cognitive assistance. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 68–81. ACM, 2014.

[7] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[8] Claudio Ragona, Fabrizio Granelli, Claudio Fiandrino, Dzmitry Kliazovich, and Pascal Bouvry. Energy-efficient computation offloading for wearable devices and smartphones in mobile cloud computing. In *Global Communications Conference (GLOBECOM), 2015 IEEE*, pages 1–6. IEEE, 2015.

[9] Amol Dhumane, Rajesh Prasad, and Jayashree Prasad. Routing issues in internet of things: A survey. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, pages 16–18, 2016.

[10] Yicong Tian and Rui Hou. An improved aomdv routing protocol for internet of things. In *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on*, pages 1–4. IEEE, 2010.

[11] C Li, C Zhao, L Zhu, H Lin, and J Li. Geographic routing protocol for vehicular ad hoc networks in city scenarios: a proposal and analysis. *International Journal of Communication Systems*, 27(12):4126–4143, 2014.

[12] Varun G Menon, PM Jogi Priya, and PM Joe Prathap. Analyzing the behavior and performance of greedy perimeter stateless routing protocol in highly dynamic mobile ad hoc networks. *Life Science Journal*, 10(2):1601–1605, 2013.

[13] Ritesh Gupta and Parimal Patel. An improved performance of greedy perimeter stateless routing protocol of vehicular adhoc network in urban realistic scenarios. 2016.

[14] Mark Berman, Jeffrey S Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61:5–23, 2014.

[15] Lide Zhang, Birjodh Tiwana, Robert P Dick, Zhiyun Qian, Z Morley Mao, Zhaoguang Wang, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2010 IEEE/ACM/IFIP International Conference on*, pages 105–114. IEEE, 2010.

[16] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[17] Gabriel Orsini, Dirk Bade, and Winfried Lamersdorf. Computing at the mobile edge: designing elastic android applications for computation offloading. In *IFIP Wireless and Mobile Networking Conference (WMNC), 2015 8th*, pages 112–119. IEEE, 2015.

[18] Lei Yang, Jiannong Cao, Hui Cheng, and Yusheng Ji. Multi-user computation partitioning for latency sensitive mobile cloud applications. *IEEE Transactions on Computers*, 64(8):2253–2266, 2015.

[19] Ke Zhang, Yuming Mao, Supeng Leng, Quanxin Zhao, Longjiang Li, Xin Peng, Li Pan, Sabita Maharjan, and Yan Zhang. Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. *IEEE Access*, 4:5896–5907, 2016.

[20] Shiqiang Wang, Rahul Urgaonkar, Murtaza Zafer, Ting He, Kevin Chan, and Kin K Leung. Dynamic service migration in mobile edge-clouds. In *IFIP Networking Conference (IFIP Networking), 2015*, pages 1–9. IEEE, 2015.

[21] Dan Andersson, Peter Elmersson, A Juntti, Z Gajic, D Karlsson, and L Fabiano. Intelligent load shedding to counteract power system instability. In *Transmission and Distribution Conference and Exposition: Latin America, 2004 IEEE/PES*, pages 570–574. IEEE, 2004.

[22] N Perumal and Aliza Che Amran. Automatic load shedding in power system. In *Power Engineering Conference, 2003. PECon 2003. Proceedings. National*, pages 211–216. IEEE, 2003.

[23] Ali Asghar Alesheikh, Hussein Helali, and HA Behroz. Web gis: technologies and its applications. In *Symposium on geospatial theory, processing and applications*, volume 15, 2002.

[24] Niraj Tolia, David G Andersen, and Mahadev Satyanarayanan. Quantifying interactive user experience on thin clients. *Computer*, 39(3):46–52, 2006.

[25] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4), 2009.

[26] Pankaj Rohal, Ruchika Dahiya, and Prashant Dahiya. Study and analysis of throughput, delay and packet delivery ratio in manet for topology based routing protocols (aodv, dsr and dsdv). *international journal for advance research in engineering and technology*, 1(2):54–58, 2013.

[27] Michael Frey, Friedrich Grose, and Mesut Gunes. Energy-aware ant routing in wireless multi-hop networks. In *Communications (ICC), 2014 IEEE International Conference on*, pages 190–196. IEEE, 2014.

[28] Yan Yu, Ramesh Govindan, and Deborah Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. 2001.

[29] Haoru Su, Zhiliang Wang, and Sunshin An. Maeb: routing protocol for iot healthcare. 2013.

[30] Gayathri Tilak Singh and Fadi M Al-Turjman. Cognitive routing for information-centric sensor networks in smart cities. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International*, pages 1124–1129. IEEE, 2014.

[31] Athina Bourdena, Constandinos X Mavromoustakis, George Kormentzas, Evangelos Pallis, George Mastorakis, and Muneer Bani Yassein. A resource intensive traffic-aware scheme using energy-aware routing in cognitive radio networks. *Future Generation Computer Systems*, 39:16–28, 2014.

[32] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[33] Spyros Gidaris and Nikos Komodakis. Object detection via a multi-region and semantic segmentation-aware cnn model. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1134–1142, 2015.

[34] Dmitrii Chemodanov, Flavio Esposito, Andrei Sukhov, Prasad Calyam, Huy Trinh, and Zakariya Oraibi. Agra: Ai-augmented geographic routing approach for iot-based incident-supporting applications. *Future Generation Computer Systems*, 2017.

[35] Andrej Cvetkovski and Mark Crovella. Hyperbolic embedding and routing for dynamic graphs. In *INFOCOM 2009*, pages 1647–1655. IEEE, 2009.

[36] Matthew A Turk and Alex P Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 586–591. IEEE, 1991.

[37] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[38] Davis E King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10(Jul):1755–1758, 2009.

[39] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

[40] Charles Perkins, Elizabeth Belding-Royer, and Samir Das. Ad hoc on-demand distance vector (aodv) routing. Technical report, 2003.

[41] Guido R Hiertz, Dee Denteneer, Sebastian Max, Rakesh Taori, Javier Cardona, Lars Berlemann, and Bernhard Walke. Ieee 802.11 s: the wlan mesh standard. *IEEE Wireless Communications*, 17(1), 2010.

[42] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14(14):527, 2008.

[43] National Oceanic and Atmospheric Organization. - Last accessed, 2016.

[44] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic routing in intermittently connected networks. *ACM SIGMOBILE mobile computing and communications review*, 7(3):19–20, 2003.

[45] Erik Kuiper and Simin Nadjm-Tehrani. Geographical routing in intermittently connected ad hoc networks. In *Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on*, pages 1690–1695. IEEE, 2008.

[46] Sahel Sahhaf, Wouter Tavernier, Didier Colle, Mario Pickavet, and Piet Demeester. Experimental validation of resilient tree-based greedy geometric routing. *Computer Networks*, 82:156–171, 2015.

[47] Simon S Lam and Chen Qian. Geographic routing in d-dimensional spaces with guaranteed delivery and low stretch. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 257–268. ACM, 2011.

[48] Michal Król, Eryk Schiller, Franck Rousseau, and Andrzej Duda. Weave: Efficient geographical routing in large-scale networks. In *EWSN*, pages 89–100, 2016.

[49] GoTenna mesh networking, 2017.