This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2018.2865661, IEEE Transactions on Multimedia

Energy-aware Mobile Edge Computing and Routing for Low-latency Visual Data Processing

Huy Trinh, Prasad Calyam, Dmitrii Chemodanov
Shizeng Yao, Qing Lei, Fan Gao, and Kannappan Palaniappan
University of Missouri-Columbia, USA;
Email: {*hntzq4, dycbt4, syyh4, qlzm3, fgyf8*} *@mail.missouri.edu, calyamp@missouri.edu, pal@missouri.edu*

***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 paper, 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 is energy-aware and improves the geographic routing baseline performance (i.e., minimizes impact of local minima) for throughput performance sustainability, while also enabling flexible policy specification. We evaluate our proposed algorithms by conducting experiments on a realistic edge and core cloud testbed in the GENI Cloud infrastructure, and recreate disaster scenes of tornado damages within simulations. Our empirical results show how MEC can provide flexibility to users who desire energy conservation over low-latency or vice versa in the visual data processing with a facial recognition application. In addition, our simulation results show that our routing approach outperforms existing solutions under diverse user preferences, node mobility and severe node failure conditions.**

***Index Terms*—Mobile Edge Computing, Policy-based Cloud Management, Energy-aware Edge Routing, Low-latency Visual Data Processing**

I. Introduction

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. The real-time requirement can imply e.g., a facial recognition application processing at a speed of 15 frames/second or ≈0.1 seconds/image [1]. 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, IoT-based application data can be handled at large scale from multiple network edge sites with on-demand computation capabilities.

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, tornadoes, wildfires, hurricanes, or man-made disasters involving terrorism, edge infrastructure may be lost. Consequently, 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] or account for first responder presence levels in disaster scenes. Typically, the identification actions needs to be achieved in real-time and through processing of a high volume of images with varying resolutions at the network edge on a limited power budget.

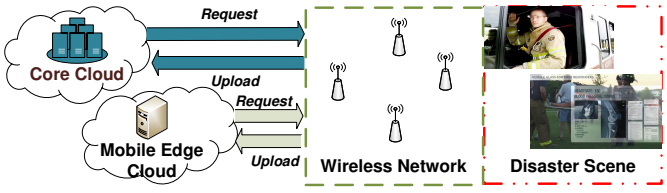


Fig. 1: Illustration of disaster scene related visual data processing by use of wireless network, mobile edge and core cloud resources to upload images and request processed images using sophisticated computer vision algorithms.

Figure 1 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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2018.2865661, IEEE Transactions on Multimedia

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]. 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. Works such as [8] have shown that cloud server offloading can save power consumption (up to 25 times) and increase processing speed (by 3X) than processing on a constrained mobile device. 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.

To address the processing-throughput versus energy-efficiency tradeoffs in MEC architectures, there is a need for flexible policy-based edge routing (i.e., a variant of geographic routing). Such an edge routing protocol should handle dynamic network situations, while also being energy-aware. For instance, the locations of mobile nodes in a Mobile Ad hoc Network (MANET) at a disaster incident scene could change frequently or static nodes could become absent within a fixed infrastructure due to power issues. This in turn can cause unpredictable topology changes [9] and create challenges for *sustainable* service continuity, as well as for maintaining routing tables. Previously proposed geographic routing protocols such as [10], [11] are stateful in nature. By maintaining node positions in a database and by updating them when events (e.g., high mobility or failure node) occur, stateful protocols can cause huge communication overheads and drain energy on power constrained IoT devices. Other stateless geographic routing protocols such as [12], [13] have been proposed that do not require maintaing routing tables, but their performance is found to degrade due to local minima i.e., they can cause infinite loops in routing.

In this paper, 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.

Our contributions. As part of paper 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 consumption 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.

To address the needs of flexible policy-based edge routing, we propose a 'Sustainable Policy-based Intelligence-Driven Edge Routing' (SPIDER) algorithm that uses machine learning techniques on satellite images to learn the geo-information about existing physical obstacles. We leverage geographic coordinates obtained via a Global Positioning System (GPS) to improve the geographic routing in terms of throughput performance sustainability in a manner that boosts baseline performance (i.e., avoids the impact of local minima). In addition, we present our SPIDER routing engine implementation whose source code is openly avaiable at [14]. Our implementation provides an edge network routing solution with flexible policy specification to handle dynamic network situations, while addressing tradeoffs in user decisions favoring either processing-throughput or energy-efficiency.

We evaluate our energy-aware and low-latency MEC framework featuring the facial recognition application and our offloading 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 [15] for the core cloud. We leverage the Android-based PowerTutor utility [16] to profile and estimate energy consumption (Metric: Joules) of our facial recognition application that is based on OpenCV [17] 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.

We evaluate our SPIDER algorithm by recreating disaster scenes within NS-3 (Network Simulator) [18] simulations that are specifically based on events during the tornado damage in Joplin areas, Missouri, USA in 2011. Our simulations feature disaster scenes and situations involving diverse user preferences, node mobility and severe node failure conditions. We leverage the 'average residual power' (Metric: Joules) measurements, which are indicative of the network lifetime in our UDP-based streaming application simulations. Our simulation results demonstrate that our SPIDER routing engine outperforms existing solutions [19], [20] and can pro-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2018.2865661, IEEE Transactions on Multimedia

vide flexibility to users who desire energy conservation over throughput performance sustainability, or vice versa in MEC environments.

Paper organization. The remainder of paper is organized as follows: Section II reviews prior related work. In Section III, 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. Section IV, we present our SPIDER algorithm with machine learning to improve geographic routing baseline performance. We present performance evaluations with realistic GENI Cloud testbed experiments, and simulations involving recreated disaster scenarios in Section VI. Section VII concludes the paper and suggests future work.

II. Related Works

Computation Offloading Decision-Making. Existing literature on computation offloading can be classified under two categories of work. First set of works such as [21], [22] 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 [23], [24], [25], [26] consider a "migration" strategy that offloads the entire application onto an edge server. Specifically, the authors in [23] create a device classification for prioritizing computation that is based on the channel and base station resource allocation status. In [24], the authors use a Markov decision process to dynamically offload computation within services. The authors in [25], [26] use Software-Defined Networking (SDN) to optimize edge (or fog) server selection as well as steer traffic. Specifically, they proposed a PRIMAL framework that uses SDN and integer quadratic programming to maximize the profit and minimize latency for the purpose of user application offloading. If offloading is not a viable option, authors in works such as [27], [28] propose "load shredding", 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.

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 [29] 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 [30], 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 [31] 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.

Energy-aware Geographic Routing for MANETs. There are works on geographic routing in MANETs such as Destination Sequenced Distance Vector (DSDV) [32]. When using DSDV, each node periodically updates its routing table with next hop information and the number of hops towards a destination, without considerations for energy efficiency. In comparison, works such as [33] propose keeping track of the network nodes' battery levels for routing decisions. To further improve energy-efficiency, more recent works such as [34], [35] propose clustering based on users' mobility and nodes' energy-consumption and selection of cluster heads to route packets toward an edge gateway. However, such cluster heads need to scan neighboring nodes' signals and store their information to form clusters, and are not inherently tolerant to high mobility and severe node failures. Cognitive routing approaches in works such as [36], [37] apply communication channels adaptation techniques to optimize energy-efficiency of the data transmission through use of metrics such as Quality of Information (QoI) and traffic awareness. All aforementioned approaches need to maintain some knowledge about the network i.e., they are stateful in the routing tables management. In comparison, our novel SPIDER algorithm is stateless and builds upon our previous work [38] that benefits from the physical obstacle knowledge derived from the satellite imagery by using deep learning-based detectors [39], [40] available at the edge. Such a knowledge facilitates throughput sustainability improvements over previous stateful geographic routing algorithms [19], [20] and stateless geographic algorithms such as [41] by using a notion of recovery for a local minimum [42]. Moreover, our SPIDER possesses energy-awareness and throughput performance sustainability attributes that are crucial for a facial recognition application in disaster response scenes involving austere MEC and networking environments.

III. Incident-Supporting Mobile Edge Computing

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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2018.2865661, IEEE Transactions on Multimedia

A. Application Background and Implementation

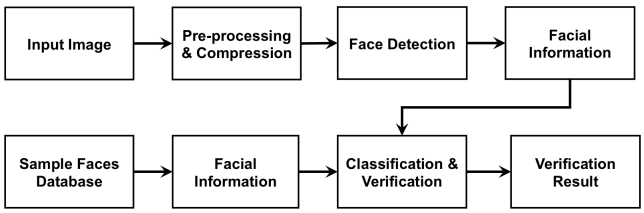


Fig. 2: Overview of stages in facial recognition for target identification.

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 2 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 using a mobile application we developed. During this pre-processing step, we compress all the input images to 1/4 of the original size i.e., we down-sample half on each dimension. Since we expect our mobile application to use relatively small datasets (on the order of tens of images), the compression has little to no effect on the classification accuracy.

After image compression, the cropped and scaled 224x224 face image, with the average face subtracted, is fed into a deep learning-based facial recognition neural network in ResNet-34 [43]. The convolutional neural network outputs an embedding vector that is used in a nearest-neighbor search to label the face image. We used the facial recognition modules in the DLib C++ toolkit for machine learning [44] as our implementation platform. DLib's face recognition (verification task) implementation uses a smaller compact version of a ResNet-34 with a few layers removed, and only half the number of filters per layer. DLib's face recognition network was trained on a pooled collection of 3 million face images of nearly 7500 individuals, from several sources including VGG [45]. The initial N-way classification or face identification network is only used as a bootstrapping stage to learn the embedding vector. In the verification task, the objective is to determine whether two face images have the same identity or represent two different people.

The DLib facial recognition architecture has a verification accuracy of 99.38% on the standard Labeled Faces in the Wild (LFW) benchmark dataset [46]. This is better in terms of performance than the Facebook DeepFace siamese network architecture trained with 4 million images of 4,000 identities and verification accuracy of 97.35% on the LFW benchmark [47], VGG-Face trained with 2.6 million images of over 2,600 people and verification accuracy of 98.95% [45], and comparable to the Google FaceNet accuracy of 99.63% that uses alignment and was trained with 200 million face images of over 8 million different identities [48].

Similar to many deep learning facial recognition architectures, the DLib implementation learns an embedding (a 128-dimensional face descriptor vector) so that congruous faces cluster together, and faces of different people are well separated after metric learning (i.e., Euclidean distance). This 128-D face descriptor is then used for facial recognition. That is after the training phase, the softmax classifier layer that outputs a weight for all person identities in the training database can be removed and the score vector from the previous layer is used as the embedding for identity verification [45]. To improve face verification performance, the embedding feature vector is usually tuned or directly learned using a triplet loss training scheme that minimizes the distance from the anchor or pivot face to positive samples, and maximizes the distances between the anchor and negative identities [48]. In DLib, a structured metric loss is used that attempts to project all the identities into non-overlapping balls of radius 0.6 using a pair-wise hinge loss in the mini-batch training set and includes hard-negative mining. The embedding network is learned once on a large collection of several million training faces from several thousand unique identities. Facial recognition with a new collection of identities then proceeds by computing the embedding for each training face image (in e.g., the disaster response face image collection) and using a scalable data structure for nearest neighbor search given a query face image.

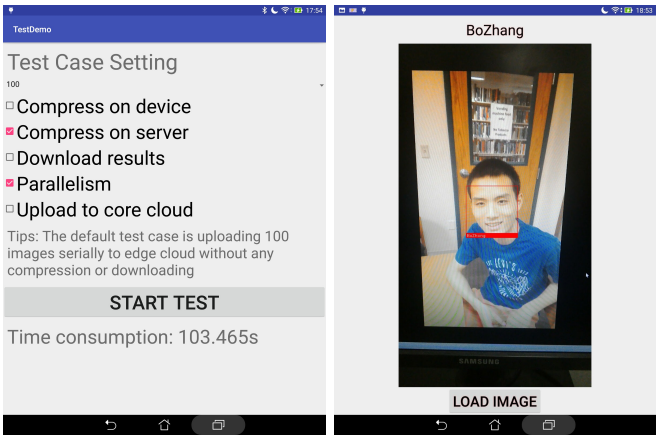


Fig. 3: Application GUI - Left Image: test options for user to select, Right Image: received result from the server side.

Figure 3 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 [17] for image management and using Python scripts that utilize Dlib [44]. To use this interface and obtain, for instance, the name of the matched image, a user can choose different computation and image transfer policies as shown in the left half of Figure 3 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 for single or multiple image uploads from one or more IoT devices simultaneously.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2018.2865661, IEEE Transactions on Multimedia

B. 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.

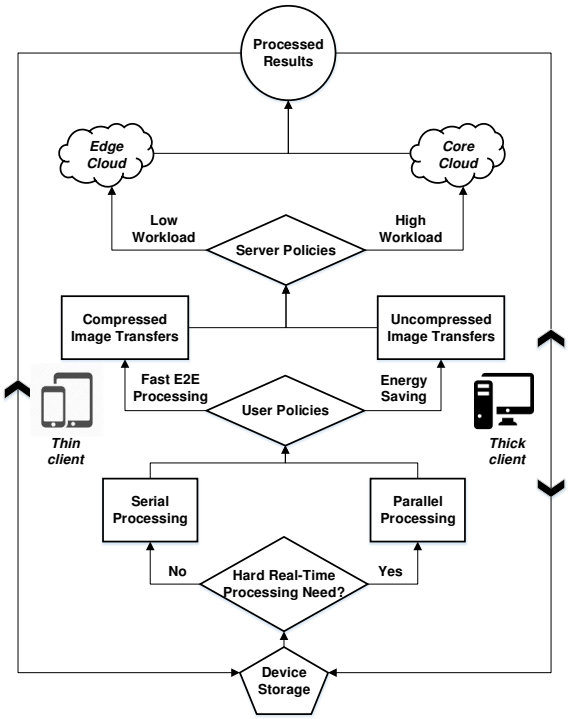


Fig. 4: Illustration showing the flexible policy-based computation offloading decision-making for low-latency visual data processing.

The application logic of the image/photo processing is displayed as Figure 4. 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 Offloading Decision-Making

Data: Load info: $resolution$, $sizeOfLoad$, $lowWorkload$
Data: Server policies: $edgeCloud$, $coreCloud$, $numOfServers$
Data: User policies: $downloadResults$, $realTimeProcess$, $saveEnergy$
Result: The efficient way to save energy and achieve low-latency processing
function $threads \leftarrow createThreads()$
  /* Create multiple threads on the mobile device for offloading load-balanced data to different servers */
  if $realTimeProcess = true$ **then**
    $threads \leftarrow parallelThreads(sizeOfLoad, numberOfServers)$
  else
    $threads \leftarrow serialThreads(sizeOfLoad, numberOfServers)$
  end
end
function $offload(threads)$
  /*Decide where to offload data for processing */
  if $saveEnergy = false$ **then**
    $compress(threads)$
  end
  if $workload(resolution, sizeOfLoad) \leq lowWorkload$ **then**
    $sendTo \leftarrow edgeCloud$
  else
    $sendTo \leftarrow coreCloud$
  end
  $send(threads, sendTo)$
end
function *main()*
  /* Decide best client configuration */
  if $downloadResults$ **then**
    use thick client
  else
    use thin client
  end
  $threads \leftarrow createThreads()$
  $offload(threads)$
end

Algorithm 1 shows our energy and latency aware steps in computation offloading. 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 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, the $createThreads()$ function ensures that the mobile device is creating multiple threads either for parallel or sequential processing based on the $realTimeProcess$ and $numberOfServers$ policies. Created threads are then used to start UDP/TCP sessions for all server

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2018.2865661, IEEE Transactions on Multimedia

instances provisioned in the core or edge cloud. Secondly, the $offload()$ function decides on whether or not threads should be compressed based on the $saveEnergy$ user's policy to trade-off between the energy consumption and latency (a.k.a. processing time). Specifically, image compression can help by reducing the data transfer time which adds to the latency at the expense of user's device energy consumption. The $offload()$ function also decides on sending data either to the edge or core cloud servers depending on the workload scale. Intuitively, 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. 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.

We remark that the workload thresholds as well as core/edge cloud location selections are specific to application/infrastructure (including user device capability) factors, and are specified via $lowWorkload$, $coreCloud$ and $edgeCloud$ policies, respectively. In the case of having multiple core/edge clouds, our algorithm can be used in conjunction with other existing schemes such as [25], [26]. Such schemes can first optimize core/edge cloud servers selection to specify $coreCloud$ and $edgeCloud$ server policies. Based on specified user/server policies, our algorithm can subsequently be executed to make final offloading decisions. In Section VI-A, we experimentally quantify image processing workload levels tailored to a facial recognition application running within an experimental core/edge cloud infrastructure.

Algorithm 1 asymptotic computational complexity. Algorithm 1 performs in a decision tree manner, i.e., it checks a number of predefined policies to decide on the offloading strategy. Due to the fact that the number of policies $\mathcal{P}$ is constant, Algorithm 1 has the following asymptotic computational complexity:

$$O(\mathcal{P}) = O(1). \tag{1}$$

From Equation 1 we can see that the Algorithm 1 complexity depends neither on the size of the workload nor on the infrastructure size. Note however that specifying some policies such as $edgeCloud$ or $coreCloud$ (optimal) servers (used in Algorithm 1) may require more complex algorithms.

IV. Energy-aware and Sustained Performance Edge Routing

Having a policy-based offloading decision scheme allows us to make flexible decisions on "*where*" to offload users' visual data processing. However, a comprehensive framework also has to address the question about "*how*" to offload users' data to deliver desired Quality of Application. In some cases of disaster incident response situations, users may prefer to have a low-latency data processing over a better energy management on power constrained IoT devices. Thus, there is a need for an edge routing algorithm that is intelligent in disaster incident scenarios and is flexible in the decision making to handle diverse 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 literature to work in challenging disaster-incident conditions involving high node mobility and severe node failures.

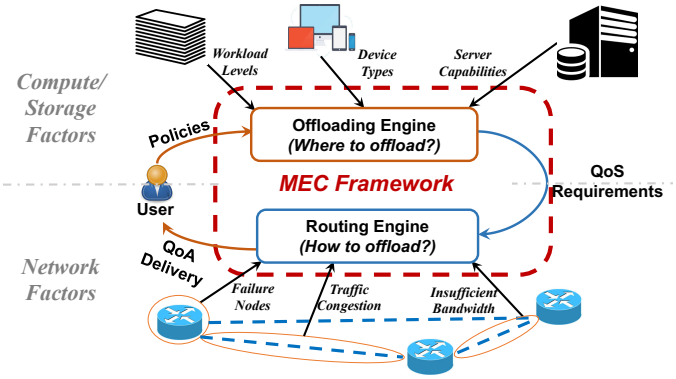


Fig. 5: Life-cycle of our MEC Framework: the relation between our offloading decision making scheme and the edge routing within our MEC framework.

A. 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 5 shows the life-cycle of our MEC framework, where at the first step users specify their policies such as thin vs. thick clients, compressed vs. non-compressed data, sequential vs. parallel processing 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 and server capabilities. Subsequently, 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 is steered taking into account various network factors such node failures, congestion and lack of bandwidth. Finally, based on the delivered QoA levels, users can modify their preferences and experiment with different sets of policies.

We remark that having a policy-based edge routing that optimizes throughput and energy efficiency trade-offs can synergistically improve IoT data offloading in a best-effort manner. Particularly, such a routing optimization could allow for a user/server policy reconsideration that further improves offloading decision-making (see Section III-B). Changing offloading policy in turn can further help optimize networking by better balancing between residual wireless network energy capacity and its throughput. However, such a best-effort optimization scheme can be still suboptimal with respect to the joint optimization of network/compute resources. Additionally, such a joint optimization can also be practically intractable due to unknown *apriori* user policies (e.g., energy efficiency of an IoT device vs. the corresponding end-to-end processing time), or due to difficulties in obtaining timely global knowledge of an infrastructure, whose wireless nodes can be subject to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2018.2865661, IEEE Transactions on Multimedia

mobility and failures (e.g., commonly occurring in the event of natural or man-made disaster incidents).

In order to satisfy network QoS requirements w.r.t user policies and various network factors (i.e., severe node failures, node mobility, traffic congestion, and insufficient bandwidth) that are common in MANETs, our proposed SPIDER algorithm details are presented in the following section.

B. SPIDER Solution Approach

As mentioned previously, we are interested not only in improving our routing throughput performance sustainability in regions affected by disaster aftermath, but also in making our solution be policy-based in order to better serve MEC user 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 neighbors, e.g., by periodically beaconing them[1]
3) each node is also aware about local obstacles' radius and location detected by the edge cloud gateway

We remark that our SPIDER solution approach has no strong assumptions on a given MANETs' topology such as unit-disk graphs or symmetric links. In addition, our SPIDER solution improves the baseline performance of the geographic routing and builds upon our previous work on Attractive, Repulsive and Pressure Greedy Forwarding (ARPGF) [38]. Similarly to ARPGF, our SPIDER solution 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 in Section IV-B1. 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.

1) SPIDER Objective: Let us consider the following model where node n forwards packet p towards destination d. In this model, node n needs to decide which neighbor should receive p to firstly progress towards d and secondly 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 node n's neighbor e with the minimum value of the following objective function:

$$f(e, d.x, d.y, \lambda) = \lambda \|\varphi(e.x, e.y, d.x, d.y)\| + (1-\lambda)\|E(e)\|, \quad (2)$$

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

where $\varphi(e.x, e.y, d.x, d.y)$ is the convex potential function of node e with respect to the destination node d that allows us to have theoretical guarantees on packets delivery with $O(3.291)$ approximation of the shortest path [38]; $E(e)$ is a residual energy at node e; 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.

Note however, that minimization of the objective function in Equation 2 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 scene related challenges.

In order to compute $\varphi(e.x, e.y, d.x, d.y)$, SPIDER needs 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 geographical locations [38]. 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 $C_j.x$ and $C_j.y$, it computes φ as following:

$$\varphi(e.x, e.y, d.x, d.y) = -\frac{1}{dist(e.x, e.y, d.x, d.y)} + \\ + \sum_{j=1}^{M} \frac{o_j(d.x, d.y)}{dist(e.x, e.y, C_j.x, C_j.y)^{\delta}} \quad (3)$$

where $dist(x1, y1, x2, y2)$ is a geographical (e.g., Euclidean) distance; δ is the attenuation order of obstacles' potential fields that has been shown empirically to give best performance when $\delta \in [1, 2]$ [38]; and $o_j(d.x, d.y)$ corresponds to the obstacle j potential intensity induced by the destination node d as following:

$$o_j(d.x, d.y) = \frac{R_j^{\delta+1}}{\delta \cdot \left(dist(C_j.x, C_j.y, d.x, d.y) + R_j\right)^2} \quad (4)$$

2) SPIDER Algorithm: Algorithm 2 outlines how each node forwards packets using either Attractive, Repulsive, or Pressure Greedy Forwarding modes. We remark that the *Attraction* mode aims to deliver packets without obstacles awareness, whereas the *Repulsive* mode aims to deliver packets with such awareness. Thus, we alternate both modes to maximize chances of proactively avoiding local minima while performing greedy forwarding of packets [38]. The *Pressure* mode was initially proposed by [42] and can be used to guarantee packet delivery by reactively recovering packets from local minima during their greedy forwarding.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2018.2865661, IEEE Transactions on Multimedia

Thus, node e starts by checking if it has the destination P_d neighbor. If not, it then checks if it has any local obstacles known, i.e., $e.\vec{C}$ and $e.\vec{R}$ are not empty (see $main()$). If so, it proceeds in *Repulsive_Forwarding()* mode. If it is not true or e faces a local minimum (i.e., it cannot find the next hop node), SPIDER proceeds in the *Attractive_Forwarding()* mode. To this end, it first temporally omits all known obstacles by setting $e.\vec{C} \leftarrow \emptyset$ and $e.\vec{R} \leftarrow \emptyset$) to avoid packets repulsion (i.e., compute $\varphi(e.x, e.y, d.x, d.y)$ without second term) (see Equation 3). Finally, if neither attractive nor repulsive forwarding modes are able to find next hop node (i.e., both are in local minima), e enters the *Pressure_Forwarding()* mode. The key idea behind this mode is to forward packets to the closest to the destination neighbor among the least visited neighbors. As a result, at some point we should be able to recover either *Attractive* or *Repulsive* modes by hitting a node n with $f(n, P_{d.x}, P_{d.y}, P_\lambda) < f(e, P_{d.x}, P_{d.y}, P_\lambda)$, where e is a node that enters the *Pressure* mode.

The key difference of the SPIDER algorithm in comparison with our previous ARPGF algorithm is that SPIDER forwards packets based on minimization of the policy-based objective function $f(e, d.x, d.y, \lambda)$ (see Equation 2). However, such flexibility has a downside as in the general case the convexity of f is not guaranteed, and hence, packets may not reach the destination d. This is due to the fact that e.g., a chosen 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 [38]. 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 (see $Feasible\_Neighbors()$ function) to deliver packets.

Algorithm 2 asymptotic computational complexity. In the worst case scenario, Algorithm 2 proceeds in all 3 modes: Attractive, Repulsive and Pressure modes. The asymptotic computational complexity of each mode is $O(k)$, where k is an average node degree. This is because each node checks the objective value f of all of its neighbors. Thus, Algorithm 2 has the following asymptotic computational complexity:

$$O(3 \cdot k) = O(k). \tag{5}$$

Note however that wireless ad-hoc networks usually have a *scale-free* nature, i.e., the average node degree k in such networks follows a power low, e.g., $P(k) \sim k^{-\gamma}$, and have strong clustering properties. As a result, k doesn't usually depend on the network size.

Algorithm 2 SPIDER

function $Repulsive\_Forwarding\ (e,\ P)$
  $f_e \leftarrow f(e, P_{d.x}, P_{d.y}, P_\lambda)$
  /* Get next hop according to φ function */
  $nextNodes \leftarrow Feasible\_Neighbors(e, P)$
  $nextAddr \leftarrow \underset{n \in nextNodes}{\operatorname{argmin}} f(n, P_{d.x}, P_{d.y}, P_\lambda)$
  if $f_e < f(n, P_{d.x}, P_{d.y}, P_\lambda)$ **then**
    return $nextAddr$
  else
    return NIL
  end
end
function $Attraction\_Forwarding(e, P)$
  e temporally sets $e.\vec{C} \leftarrow \emptyset$ and $e.\vec{R} \leftarrow \emptyset$ to omit P repulsion
  $f_e \leftarrow f(e, P_{d.x}, P_{d.y}, P_\lambda)$
  //Get next hop according to φ function with only first term
  $nextNodes \leftarrow Feasible\_Neighbors(e, P)$
  $nextAddr \leftarrow \underset{n \in nextNodes}{\operatorname{argmin}} f(n, P_{d.x}, P_{d.y}, P_\lambda)$
  if $f_e < f(n, P_{d.x}, P_{d.y}, P_\lambda)$ **then**
    return $nextAddr$
  else
    return NIL
  end
end
function $Pressure\_Forwarding(e, P)$
  //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 $P_{visits(n)} == visits_{min}\}$
  $P_{visits(n)} \leftarrow P_{visits(n)} + 1$
  $nextAddr \leftarrow \underset{n \in Nbrs(e)}{\operatorname{argmin}} f(n, P_{d.x}, P_{d.y}, P_\lambda)$
  return $nextAddr$
end function $Feasible\_Neighbors(e, P)$
  //Get set of next hop according toward destination
  $\varphi_e \leftarrow \varphi(e.x, e.y, P_{d.x}, P_{d.y})$
  foreach $n \in Nbrs(e)$ **do**
    $\varphi_n \leftarrow \varphi(n.x, n.y, P_{d.x}, P_{d.y})$
    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 $P_d \in Nbrs(e)$ **then**
    $nextAddr \leftarrow P_d$
  else
    $nextAddr \leftarrow NIL$
    if $e.\vec{C} \notin \emptyset$ *and* $e.\vec{R} \notin \emptyset$ **then**
      $nextAddr \leftarrow Repulsive\_Forwarding(e, P)$
    end
    if $nextAddr = NIL$ **then**
      $nextAddr \leftarrow Attraction\_Forwarding(e, P)$
    end
    if $nextAddr = NIL$ **then**
      $nextAddr \leftarrow Pressure\_Forwarding(e, P)$
    end
    $send(P, nextAddr)$
  end
end

V. MEC Framework Architecture

In this section, we describe our MEC framework architecture that combines both our offloading decision making scheme and our SPIDER solution approach in disaster scenarios featuring: (a) a facial recognition application to improve the visual situational awareness, and (b) deep learning to improve SPIDER performance. Our combined MEC Framework architecture shown in Figure 6 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 services in the Core Cloud to coordinate our SPIDER routing in MANET environments.

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 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 com-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2018.2865661, IEEE Transactions on Multimedia

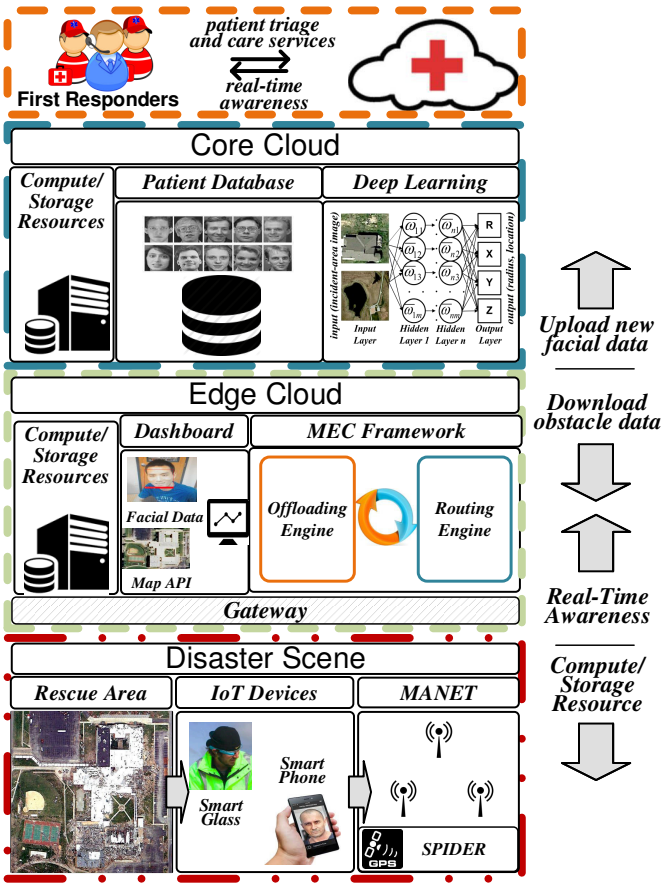


Fig. 6: Illustration of our MEC Framework architecture that consists of three main logical components: the MEC Framework itself; its offloading engine that interacts with user IoT and dashboard; and the MEC routing engine that coordinates our SPIDER routing in MANET powered by deep learning in the Core Cloud.

pute/storage factors such as workload levels, device types, and server capabilities. 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, and insufficient bandwidth.

MEC Offloading Engine, User Dashboard and IoT. The IoT devices such as security cameras, civilian smart phones, and aerial perspectives collect patients' visual data that need compute and storage resources for their processing available at the hybrid Edge/Core Cloud. To augment user IoT devices with demanded compute/storage resources, we use our "Offloading Engine". Based on specified user policies, this engine decides on suitable offloading strategy, e.g., should we offload IoT data processing to the Edge or Core Cloud. In our architecture, we use a dashboard to allow users specify their offloading policies, i.e., compressed/non-compressed, thin/thick client, parallel/sequential processing, and energy-efficient offloading versus low-latency processing. Finally, upon obtaining new processed data, users may also wish to use a dashboard for searching the data sets on the map or share them via the core cloud storage for public access, e.g., new facial data can be uploaded to the public patient database for matching and verification to ease e.g., finding lost people.

MEC Routing Engine, SPIDER and Deep Learning. Upon receiving network QoS guidance as well as user policies from the offloading engine (see Section IV-A), 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 later use within our SPIDER objective function f (see Equation 2). 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 algorithm in this area. We remark that such obstacles can cause packet drops due to lack of wireless connectivity in their proximity. After the rescue area has been learned, our "Routing Engine" propagates information about discovered geographic obstacles through the gateway to the MANET nodes in the disaster-incident scene. Further, node n in MANET stores only those obstacles' information that locate within two radius proximity from it [38]. We remark that n needs this information to compute its $\varphi(n)$ (see Equation 3).

To learn information about geographic obstacles (either proactively or reactively), our Routing Engine can use deep learning detectors and a publicly available map API that contains satellite imagery of the disaster-incident scene (not necessarily the newest one). As and when a map API is available at the user dashboard, our routing engine may detect obstacles even without using the Core Cloud deep learning service, i.e., offline. For instance, the You Only Look Once (YOLO) [39] deep-learning detector labels objects in the image using only a single neural network comprised of 26 layers which makes it easier to run on the resource-constrained Edge Cloud. At the same time, it can have worse performance compared to more sophisticated (i.e., more resource eager) deep learning detectors [39], [40]. 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 [49] to enhance performance of detectors in the future. We remark that finding the best (i.e., the most accurate) approach for obstacle detection on a given satellite map can further enhance our routing engine performance, however such an investigation is beyond the scope of this paper.

VI. Performance Evaluation

In this section, we first evaluate our energy-aware and low-latency MEC framework using various policies such as low-to-high workloads, thin or thick clients, and sequential or parallel processing at both core and edge clouds. Following this, we evaluate our SPIDER algorithm performance through extensive simulations using diverse policies, realistic disaster scenes of damaged areas under challenging high node mobility and severe node failure conditions.

A. Visual Data Processing Evaluation

Experimental Settings. Figure 7 shows our experimental testbed setup where we use a local U. of Missouri server

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2018.2865661, IEEE Transactions on Multimedia

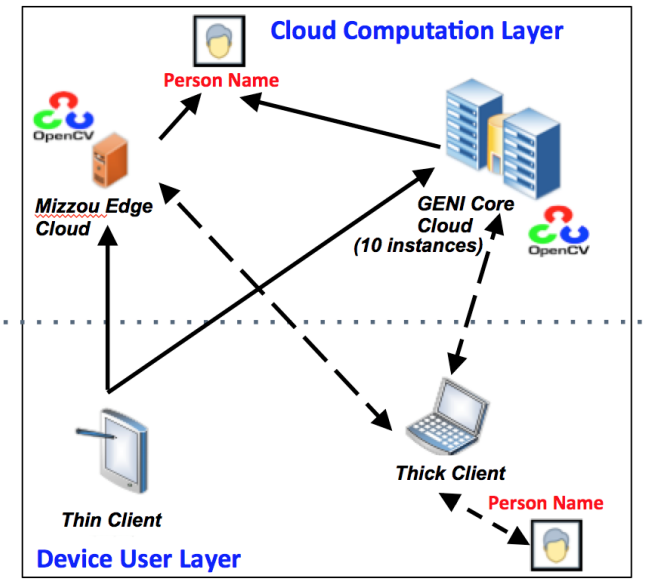


Fig. 7: Testbed edge cloud is a server on Mizzou campus and our core cloud includes 10 GENI server instances at New York University (NYU) campus.

resource [15] 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 mobile device that runs the facial recognition application described in Section III.

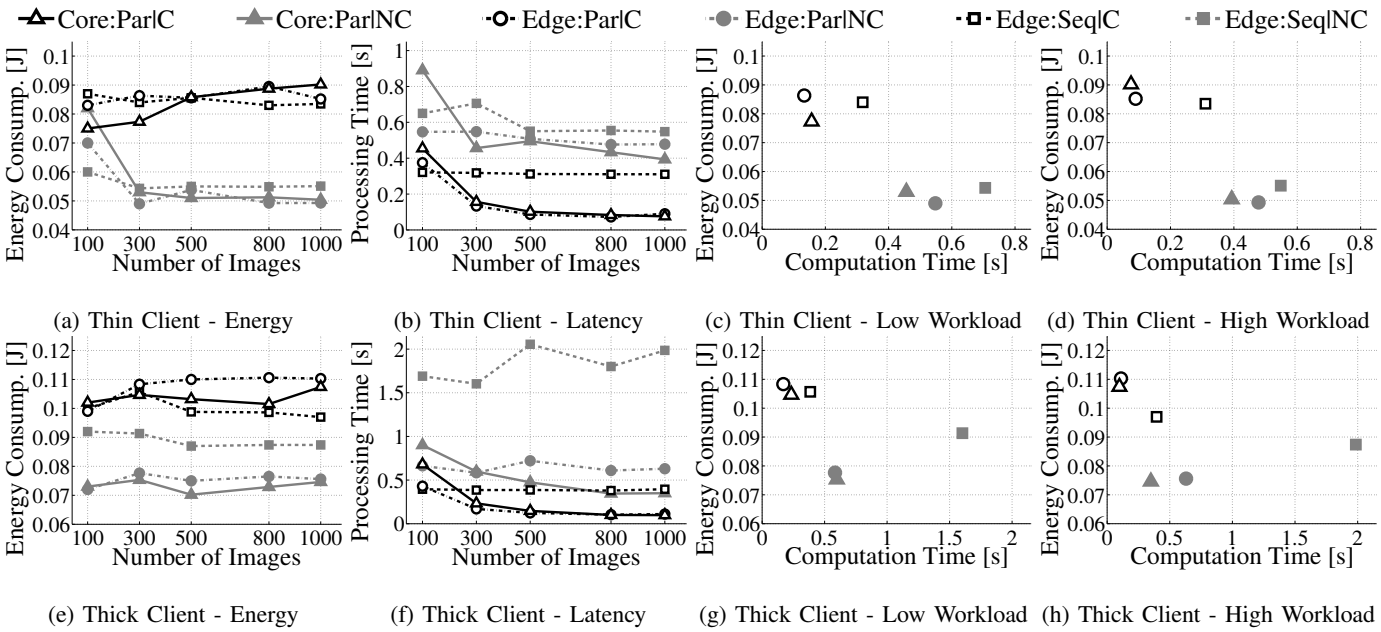
Comparison methods and metrics. We compare cases where *thin* client or *thick* client configurations are used. Particularly, our thin client configuration at a user assumes all of the processed images are stored and viewed at a remote cloud resource, whereas a thick client configuration assumes processed 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 7. 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 PowerTutor utility [16] 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). 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.

B. Discussion

Policy-based Optimization. We start our MEC framework evaluation by discussing its optimal policy sets (defined by a combination of decision parameters in Figure 4) that cover diverse user's demands and input data scale. Based on energy consumption results in Figures $8a$ and $8e$, 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 $8b$ and $8f$ 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., $\leq$ 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.

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 $8c$ and $8g$ of a low workload scale (i.e., when processing $\approx$300 images), we can see how *parallel* processing 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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2018.2865661, IEEE Transactions on Multimedia



(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

Fig. 8: 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)

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 ≈1000 images). Particularly, observing Figures $8d$ and $8h$, 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.

C. Edge Routing Evaluation

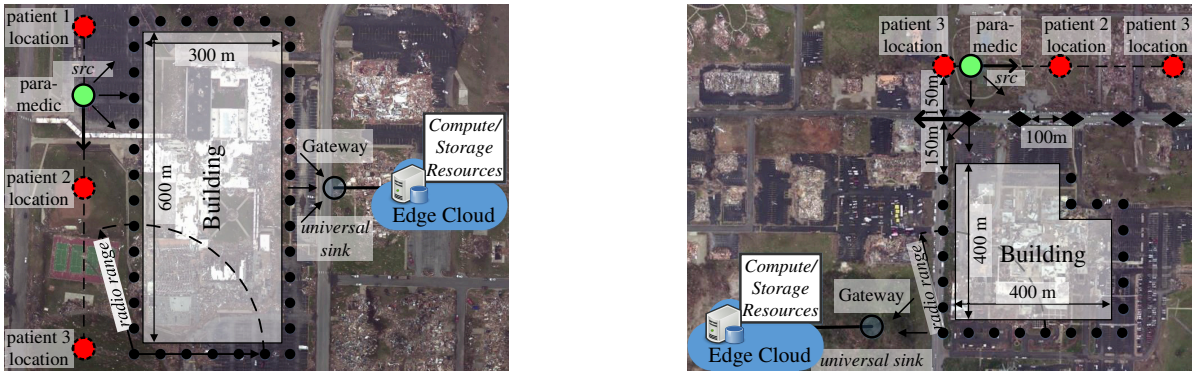
To evaluate our SPIDER performance, we use realistic disaster-incident scenarios with severe node failures and high node mobility in NS-3. We then compare its performance with the flexible stateless greedy routing GEAR protocol [41]. We also compare our SPIDER with common stateful ad-hoc routing solutions: the known reactive Ad-Hoc On Demand Distance Vector (AODV) protocol [19]; and the Hybrid Wireless Mesh Network (HWMP) protocol 802.11s standard [20] that combines reactive (by using AODV) as well as proactive routing (by using spanning trees).

Simulation Settings. For our evaluation, we have implemented our SPIDER algorithm in NS-3 simulator [18]. We also use realistic disaster-affected scenes (see Figures 9 (a) and (b)) of damaged areas due to the tornado that hit the Joplin High School and Joplin Hospital buildings in Joplin, Missouri in 2011. We obtained the disaster affected scenes information from the available satellite imagery maps showing tornado effects [50]. Using the scene information, we evaluate the performance of stateless greedy forwarding algorithms under mobility and severe node failure conditions. We assume that the information regarding damaged buildings ($i.e.$, their center coordinates and radius) are provided from the edge cloud through a Gateway using satellite imagery of the rescue areas and deep learning obstacle detectors (see Section V) in the core cloud.

In our disaster-incident experiment scenario, a paramedic acts as a source sending data to the gateway over a MANETs. Video streams gathered on-site are sent over a UDP session to the edge cloud for further data processing in conjunction with the 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. We remark that such IoT devices do not have sufficient compute and storage resources to perform facial recognition functions locally and have to send their captured streams to the hybrid edge/core cloud. The paramedic stays for 3 minutes at each patient location and moves at a jogging speed ($\approx$ 6 mph) between these locations. The simulation is designed to cause a geographical routing to face an abandoned wireless coverage zone 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 9a), nodes around an obstacle can fail for the next 30 seconds due to the possibility of an intermittently available power supply, or due to a physical damage near the disaster scene. Their failure probability is sampled from the interval $[5\%, 50\%]$, $i.e.$, from low to severe

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2018.2865661, IEEE Transactions on Multimedia



(a) (b)

Fig. 9: First scenario (a), we evaluate our approach under severe failures, and in the second scenario (b), we evaluate our approach under high mobility.

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 for such node failure scenarious any "store and forward" solutions can be inadequate [51], [52].

We then evaluate impact of the node mobility in the second simulation scenario (see Figure 9b), where paramedics can communicate with the edge gateway only through movement on the road vehicles with the speed varying from ≈ 10 mph (low) to ≈ 40 mph (high). Note that under such mobility conditions, any stateful routing solution ($i.e.$, which relies on the network topology knowledge such as spanning trees) will exhibit poor performance [53].

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 I summarizes all of our simulation details.

TABLE I: Simulation Environment Settings

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:	≈ 0.033 Hz	Technology:	802.11g/s
Node failure probability:	0.05 - 0.5	Modulation:	OFDM
Mobile nodes speed:	10 - 40 mph	Data rate:	54 Mbps
Time at each location:	180 sec	**Transport/App layers:**	
Src speed:	6 mph	Transport protocol:	UDP
Simulation time:	720 - 780 s	Payload:	1448 bytes
Beaconing frequency:	1 - 4 Hz	Application bit rate:	5 Mbps

Comparison methods and metrics. In our realistic simulation, we assess routing performance of both SPIDER and GEAR by experimentation with different policies. For example, we try setting $\lambda = 1$ to obtain the best latency and throughput results, $\lambda = 0$ to achieve the best energy-efficiency, and $\lambda \in \{0.25, 0.50, 0.75\}$ to get the balanced solution in terms of both the energy-efficiency and the resulting throughput and latency.

We use a simple energy model in which every node starts with the same initial energy budget (i.e., 1000 Joules), and consumes one unit of energy for either transmitting or receiving a packet. We then compute the network *residual energy* in Joules by averaging the residual energy of all nodes' batteries in the network. We also measure the application level *throughput* in Mbps of the paramedic video streaming to the edge cloud.

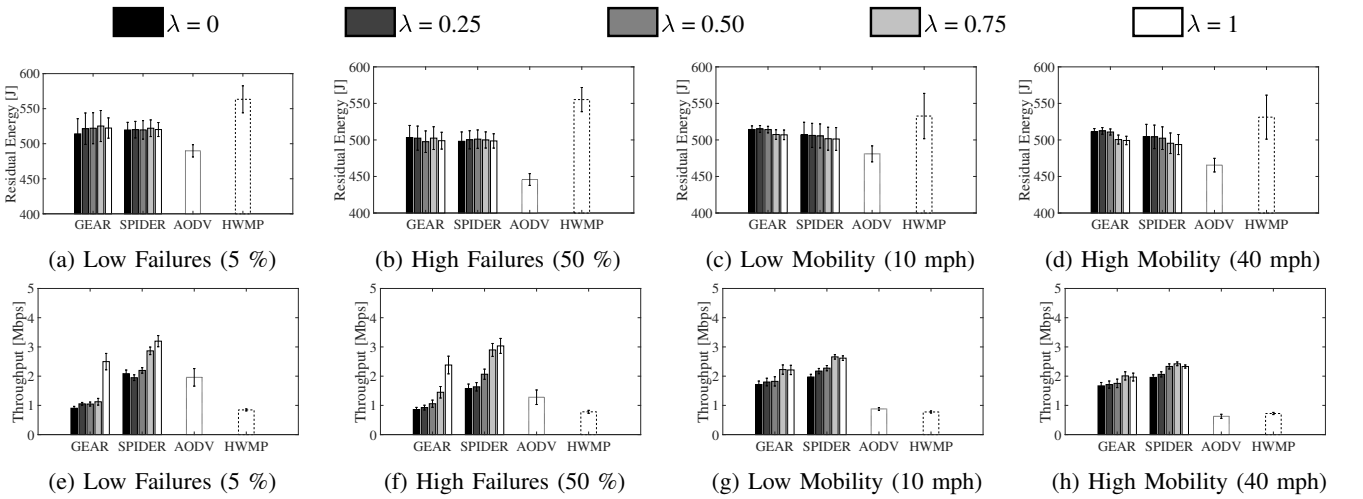
D. Discussion

SPIDER & HWMP are Pareto-Optimal Routing Strategies. Figure 10 shows how for every tested λ parameter in Equation 2 our SPIDER outperforms related GEAR and AODV protocols by demonstrating the highest application throughput level, and at the same time it is comparable or better in terms of energy-efficiency. However, HWMP shows better energy efficiency than SPIDER due to its ability to use spanning trees for minimizing the number of control messages. However, these spanning trees degrade HWMP performance under severe node failures or high node mobility resulting in the lowest application throughput w.r.t. other protocols. Thus, we conclude that both SPIDER and HWMP are Pareto-Optimal routing strategies in MANETs, i.e., these routing strategies have no alternative strategies that make any one preference criterion (e.g., energy or low-latency) better off without making at least one preference criterion worse off.

Too much emphasis on a single hop energy-efficiency can lead to the bad network energy-efficiency. While observing Figure 10, we notice how $\lambda = 0$ does not lead to the best network energy-efficiency of both GEAR and SPIDER all the time. For example, Figures 10a, 10b and 10d show how both GEAR and SPIDER achieve the best network energy-efficiency with $\lambda = 0.75$, $\lambda = 0.5$ and $\lambda = 0.25$, respectively. This result is due to the fact that too much emphasis on an energy-efficiency for a single hop forwarding of packets can force them to traverse longer paths. As a result, routing of these packets consumes more energy from the entire network. Thus, we recommend using $\lambda = 0.25$ for the routing energy-efficiency and $\lambda = 1$ to achieve the highest application level throughput as well as a low-latency processing. Alternatively, one can consider changing λ policy in real-time to dynamically adapt for various disaster-incident scenarios.

SPIDER improves routing sustainability in MANETs. Due to additional geographic knowledge of the rescue area (along with geographic coordinates), our SPIDER achieves

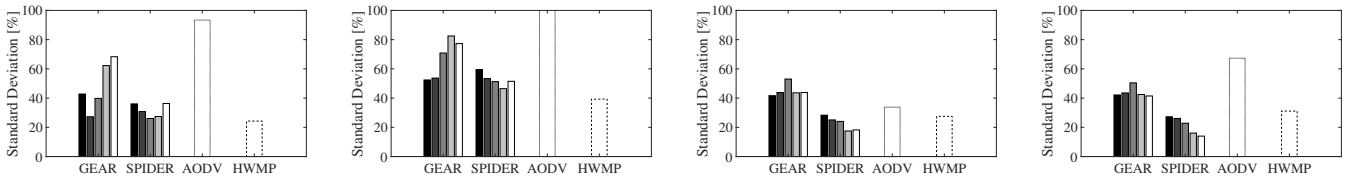
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2018.2865661, IEEE Transactions on Multimedia



(a) Low Failures (5 %) (b) High Failures (50 %) (c) Low Mobility (10 mph) (d) High Mobility (40 mph)

(e) Low Failures (5 %) (f) High Failures (50 %) (g) Low Mobility (10 mph) (h) High Mobility (40 mph)

Fig. 10: The residual network energy (*first row*) and application level throughput (*second row*) with 95% confident interval results under low (a, e) and high (c, f) node failures and in presence of the low (c, g) and high (d,h) node mobility using different routing policies λ.



(a) Low Failures (5 %) (b) High Failures (50 %) (c) Low Mobility (10 mph) (d) High Mobility (40 mph)

Fig. 11: Application level throughput sustainability (i.e., standard deviation) results under low (a) and high (b) node failures and in presence of the low (c) and high (d) node mobility using different routing policies λ.

the highest throughput performance sustainability as shown in Figure 10. This is because – when entering zones with abandoned wireless network coverage, common geographic routing approaches (e.g., GEAR) cannot forward packets using only geographic coordinates; this problem is known in optimization literature as a local minimum problem. As a result, GEAR enters the recovery mode and uses planarization which, in turn, can significantly stretch paths. However, stretching paths reduces application level throughput and increases packets' latency which, in turn, leads to a poor quality of transferred video streams in a facial recognition or any other disaster response application.

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. This could cause service outages or frequent disconnections. Recent solutions in stateful geographic routing literature can help cope with some of these disaster incident challenges [54], [55]. For example, recent geographic routing solutions have shown promising results under severe node failures [55]. However, we found no geographic routing algorithms that can cope with both severe node failures and high node mobility conditions.

Based on the above results, we conclude that our SPIDER algorithm improves routing sustainability, exhibits energy awareness, and enhances quality of user's applications – due to its knowledge of geographic obstacles located within the rescue area, which in most cases allows local minima avoidance by using its *repulsion* forwarding mode.

VII. Conclusion and Future Work

In this paper, 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 processing in MANETs to trade-off between energy-awareness and low-latency data transferring to an edge cloud gateway within a damaged infrastructure area. Specifically, we presented our Sustainable Policy-based Intelligence Driven Edge Routing (SPIDER) algorithm that builds upon recent advances in the geographic routing area. To improve its baseline geographic routing performance, SPIDER uses addi-

tional geographic knowledge that we obtain from the publicly available satellite imagery of the rescue area and from its use of deep learning detectors in a core cloud. To balance between energy-awareness and low-latency data transferring in a best-effort manner, our SPIDER algorithm used a tunable objective function. Considering a variety cases of actual disaster incident related scenarios, we have shown how our SPIDER algorithm 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).

As part of future work, practical routing protocols with load balancing can be implemented to allow parallel processing in cases where there are multiple servers available in a MEC environment. In addition, interference can be handled in wireless network when caused by high user activity in MANETs using channels adaptation techniques that: (a) minimize the chance of packet drops, and (b) enhance energy-efficiency within IoT devices.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation (NSF) under Award Number: CNS-1647182. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the NSF.

REFERENCES

[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*, pp. 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 co-ordination. In *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual*, pp. 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*, pp. 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*, pp. 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*, pp. 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*, pp. 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, pp. 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*, pp. 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. *Int. J. Scientific Research in Computer Science, Engineering and Information Technology*, Vol. 1, No. 1, pp. 24-29, 2016.

[14] Openly accessible Source Code Repository. Spider (sustainable policy-based intelligence driven edge routing) routing engine. https://github.com/huytrinh93/SPIDER, Last Accessed in August 2018.

[15] 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.

[16] 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*, pp. 105–114. IEEE, 2010.

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

[18] 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.

[19] Charles Perkins, Elizabeth Belding-Royer, and Samir Das. Ad hoc on-demand distance vector (aodv) routing. In *Proc. IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[20] 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.

[21] 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*, pp. 112–119. IEEE, 2015.

[22] 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.

[23] 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.

[24] 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*, pp. 1–9. IEEE, 2015.

[25] Xiang Sun and Nirwan Ansari. EdgeIoT: Mobile Edge Computing for Internet of Things *IEEE Communications Magazine*, volume 54, pp. 22–29, 2016.

[26] Xiang Sun and Nirwan Ansari. PRIMAL: PRofIt Maximization Avatar pLacement for Mobile Edge Computing. In *IEEE International Conference on Communications (ICC)*, pp. 1–6, 2016.

[27] 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*, pp. 570–574. IEEE, 2004.

[28] N Perumal and Aliza Che Amran. Automatic load shedding in power system. In *Proc. of IEEE Power Engineering Conference (PECon)*, pp. 211–216, 2003.

[29] 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.

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

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

[32] 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.

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

[34] Haoru Su, Zhiliang Wang, and Sunshin An. Maeb: routing protocol for iot healthcare. *Scientific Research Publishing*, 2013.

[35] Mian Ahmad Jan, Priyadarsi Nanda, Xiangjian He, and Ren Ping Liu. A sybil attack detection scheme for a forest wildfire monitoring application. *Future Generation Computer Systems*, 80(Supplement C), pp. 613 – 626, 2018.

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

[37] 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.

[38] 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.

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

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

[41] Yan Yu, Ramesh Govindan, and Deborah Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. *Technical report ucla/csd-tr-01-0023, UCLA Computer Science Department*, 2001.

[42] Andrej Cvetkovski and Mark Crovella. Hyperbolic embedding and routing for dynamic graphs. In *Proc. of IEEE INFOCOM 2009*, pp. 1647–1655, 2009.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun Deep Residual Learning for Image Recognition In *IEEE Computer Vision and Pattern Recognition*, pp. 770-778, 2016.

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

[45] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep Face Recognition *British Machine Vision Conference*, 2015.

[46] Gary B. Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. *Workshop on faces in 'Real-Life' Images: detection, alignment, and recognition*, 2008.

[47] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *IEEE Computer Vision and Pattern Recognition*, pp. 1701-1708, 2014.

[48] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *IEEE Computer Vision and Pattern Recognition*, pp. 815-823, 2015.

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

[50] National Oceanic and Atmospheric Organization. - Last accessed in August 2018.

[51] 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.

[52] 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*, pp. 1690–1695. IEEE, 2008.

[53] 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.

[54] 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*, pp. 257–268, 2011.

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

Huy Trinh received the BS degree in Computer Science at University of Missouri, Columbia, USA, in 2015. He is currently a Graduate Research Assistant working toward the M.S degree in the Department of Electrical Engineering and Computer Science, University of Missouri, Columbia, USA. His research interests include image processing, cloud computing and networks.

Prasad Calyam received his MS and PhD degrees from the Department of Electrical and Computer Engineering at The Ohio State University in 2002 and 2007, respectively. He is currently an Associate Professor in the Department of Electrical Engineering and Computer Science at University of Missouri-Columbia. His current research interests include distributed and cloud computing, computer networking, and cyber security. He is a Senior Member of IEEE.

Dmitrii Chemodanov received his MS degree from the Department of Computer Science at Samara State Aerospace University, Russia in 2014. He is currently a PhD student in the Department of Electrical Engineering and Computer Science at University of Missouri-Columbia. His current research interests include distributed and cloud computing, network and service management, and peer-to-peer networks.

Shizeng Yao received his B.S. degree from the Department of Electrical Engineering at Northeastern University, China in 2012, and received his M.S. degree from the Department of Electrical and Computer Engineering at University of Missouri-Columbia, USA, in 2015. He is currently a Graduate Research Assistant working toward his Ph.D. degree in the Department of Electrical Engineering and Computer Science at University of Missouri-Columbia. His research interests include digital image processing, computer vision, and multi-view stereo reconstruction.

Qing Lei received her Bachelor Degree of Engineering in Computer Science and Technology at Shanghai University in 2015. She is currently pursuing her MS degree in Electrical Engineering and Computer Science and working for Computer Graphics and Image Understanding Lab at the University of Missouri Columbia. Her current research interests include image processing, computer vision and computer graphics.

Fan Gao received the B.S. degree in Software Engineering from Northeastern University, China, in 2017. He is currently pursuing his M.S. degree in Electrical Engineering and Computer Science at the University of Missouri-Columbia. He is a Graduate Research Assistant in Department of Computer Science, his research interests include computer vision, image processing and machine learning.

Kannappan Palaniappan received his PhD from the University of Illinois at Urbana-Champaign, and MS and BS degrees in Systems Design Engineering from the University of Waterloo, Canada. He is a faculty member in Electrical Engineering and Computer Science Department at the University of Missouri, where he directs the Computational Imaging and VisAnalysis Lab and helped establish the NASA Center of Excellence in Remote Sensing. At NASA Goddard Space Flight Center he co- invented the Interactive Image SpreadSheet for visualizing large multispectral imagery and deformable cloud motion analysis. His research is at the synergistic intersection of image and video big data, computer vision, high performance computing and artificial intelligence to understand, quantify and model physical processes with applications to biomedical, space and defense imaging. Recent multi- disciplinary contributions range across orders of scale from sub-cellular microscopy at the molecular level to aerial and satellite remote sensing imaging at the macro level. In 2014 his team won first place at the IEEE Computer Vision and Pattern Recognition Change Detection Workshop video analytics challenge. He has received several notable awards including the William T. Kemper Fellowship for Teaching Excellence at the University of Missouri, ASEE Air Force and Boeing Welliver Summer Faculty Fellowships, the NASA Public Service Medal for pioneering contributions in data science for understanding petabyte-sized archives, and the first US National Academies Jefferson Science Fellowship from the state of Missouri. He is a member of the Editorial Board of the IEEE Transactions on Image Processing. He is a Senior Member of IEEE.