Load Profiling and Migration for Effective Cyber Foraging in Disaster Scenarios with FORMICA

Agnese V. Ventrella*§ Flavio Esposito* L. Alfredo Grieco*‡
* Computer Science Department, Saint Louis University, USA
{agnese.ventrella, flavio.esposito}@slu.edu
§Department of Electrical and Information Engineering, Politecnico di Bari, Italy
afredo.grieco@poliba.it
‡CNIT, Consorzio Nazionale Interuniversitario per le Telecomunicazioni

***Abstract*—Cyber foraging techniques have been proposed in edge computing to support resource-intensive and latency-sensitive mobile applications. In a natural or man-made disaster scenario, all cyber foraging challenges are exacerbated by two problems: edge nodes are scarce and hence easily overloaded and failures are common due to the ad-hoc hostile conditions.**

In this paper, we study the use of efficient load profiling and migration strategies to mitigate such problems. In particular, we propose FORMICA, an architecture for cyber foraging orchestration, whose goal is to minimize the completion time of a set of jobs offloaded from mobile devices. Existing service offloading solutions are mainly concerned with outsourcing a job out of the mobile responsibility. Our architecture supports both mobile-based offloading and backend-driven onloading $i.e.$, the offloading decision is taken by the edge infrastructure and not by the mobile node. FORMICA leverages Gelenbe networks to estimate the load profile of each node of the edge computing infrastructure to make proactive load profiling decisions. Our evaluation on a proof-of-concept implementation shows the benefits of our policy-based architecture in several (challenged disaster) scenarios but its applicability is broad to other IoT-based latency-sensitive applications.

I. Introduction

Mobile applications that serve the needs of disaster incident response generate large dataset and demand large computational resource access. Such datasets are usually collected in real-time at the disaster scenes using different Internet of Things (IoT) devices. Examples of such devices are wearable heads-up devices, Unmanned Aerial Vehicles equipped with sensors, cameras, or civilian tablets or smartphones [1], [2]. To enable immediate feedback to first responders, crucial for survivors' rescue, IoT devices today could benefit from the mobile edge computing paradigm [1], [3]. It is inefficient to marshal all data collected by a bundle of sensors to the cloud for processing and analysis; doing so requires a great deal of bandwidth and all the back-and-forth communication between the sensors and the cloud, and can negatively impact performance. In this paradigm, much of the processing takes place at the edge of the network (as opposed to taking placr in the core of the network as in the Cloud Computing paradigm). This (distributed) approach is growing in popularity due to the latency-critical application needs of modern applications, in modern telecommunication infrastructures such as 5G, healthcare, disaster responsiveness or video gaming industries, to name a few. Such technology is often integrated with network virtualization and Software-Defined Infrastructures that dynamically provide on-demand access to networking, computation and storage resource, wherever available.

One of the most important mechanisms in edge computing is cyber foraging: processes from mobile resources delegate computations or code to (compile or) execute to servers within the edge computing infrastructure [4]. A particular case of cyber foraging is also known as offloading. The term offloading (restrictively) implies that the cyber foraging mechanism is orchestrated by mobile devices. When the cyber foraging decision is managed instead by a process within the infrastructure, the cyber foraging process is called "onloading". We argue that (mobile) device-driven offloading and back-end driven should merely be policies of the cyber foraging mechanism. The overwhelming majority of existing solutions for cyber foraging focus on offloading [2]. As already floated in recent work [5], [6], we instead believe that backend-driven onloading is a wiser (fast and more scalable) alternative to mobile devices-driven offloading. Not only because mobile devices have resource constraints and should not be overloaded with additional decisions, but because resources at the edge of the network are programmable and hence they can be better managed.

To this end, in this paper we propose FORMICA (FORaging and MIgration Cyber Architecture), a policy-based architecture for cyber foraging and migration management. [1] By policies we mean variant aspects of the cyber foraging mechanisms and include job migration, migration (of nodes and links, $i.e.$, routing policies). With FORMICA, mobile devices simply offload their jobs to the closest edge computing node, that in turn orchestrates their execution within a software-defined infrastructure (not necessarily an OpenFlow-based SDN controller).

FORMICA is designed to be resilient to challenged edge network, such as those within a disaster scenario, and at its core is built with two main principles in mind: (i) stochastic

978-1-5386-4633-5/18/$31.00

[1] Formica in Italian means ant. Formicidae (ants) spend most of their life foraging for resources.

job runtime estimation for effective job migration across edge severs and (ii) load profiling, a more general and intelligent case of load balancing. The stochastic job runtime estimation is obtained modeling the number of offloaded jobs with a network of queues (Gelenbe Network), while the load is profiled keeping track of several network parameters and current conditions. To establish the practicality of our approach, we built a proof-of-concept prototype, whose C++ code is freely available at [7]. We also analyze the cyber foraging policy tradeoff (e.g., onloading vs offloading) with a simulations campaign. Among our simulations results, we found that back-end driven onloading is more effective $i.e.$, the average job completion time is always lower than in mobile-driven offloading.

The rest of the paper is organized as follows: we formulate the effective cyber foraging problem and discuss the sketch of our solution in Section II-A. In Section III we discuss our FORMICA architecture, its prototype core components; in Section III-A we describe the core idea behind the Gelenbe network model that FORMICA uses for its job queue estimation. In Section IV we discuss the proactive job migration strategies implemented by the migration manager, the other core component of the FORMICA architecture. In Section V-B we present our simulation results and in Section VI the related work. Finally, in Section VII we conclude our work.

II. The Effective Cyber-Foraging Problem in Natural or Man-made Disaster

Natural or man-made disaster scenarios are hostile environments that can experience frequently connectivity loss because of infrastructure problems. In this section we define the Effective Cyber-Foraging Problem and we sketch our solution.

A. Problem Definition

Cyber foraging techniques have been proposed to support these scenarios, by allowing the offloading of (Lightwave) Virtual Machines or code otherwise running on a mobile node, to nearby surrogate machines, often called cloudlets [8]. Usually, cloudlets are close to the mobile device and are reachable via a single-hop network. This ensures low-latency connections. Two problems emerge in this context: (i) edge nodes are scarse and hence easily overloaded; (ii) failures are common due to hostile conditions. Those problems often lead to unacceptable delays and significant losses, with a subsequent increase of average job completion time. In this paper, we address these challenges.

***Problem** 2.1: Given a set of (first responder) devices ready to offload a set of computationally intensive tasks ($i.e.$ jobs) on an edge computing infrastructure, we define the Effective Cyber-Foraging Problem as the edge infrastructure management problem minimizing the average completion time of an offloaded set of tasks by effectively orchestrating the load on the programmable edge computing infrastructure.*

Examples of tasks to be offloaded in a disaster scenario are preprocessing of images captured by a fleet of drones, looking for survivals, or patient health records or images captured by first responder mobile devices for identification and lost person matching. By programmable edge computing load orchestration instead we mean: (a) enforcing a given load profile on edge servers, and (b) migrate the tasks whose expected running time is high to an edge server that is (probabilistically or deterministically) most likely to complete it within a shorter time, despite the potential infrastructure failures. By load profiling instead we generalize the classical load balancing notion. A set of servers with balanced load have a very specific profile that has identical target load. Load balancing techniques are inappropriate when severe failure to the infrastructure may occur, or when edge servers (installed by first-responders) have uneven physical CPU capacities.

To solve Problem 2.1, two additional challenges need to be considered: when is it appropriate to offload (both from the mobile device to an edge computing server and among edge servers) and where should a task be migrated?

B. Our Solution: Proactive Job Orchestration

To trigger the migration of a task, we use a proactive, self-adjusting adaptation mechanism: a task offloaded to an edge computing node (i.e. server) migrates, if convenient, to another edge computing server, when its hosting node reaches a threshold. Such threshold is set to be the average number of offloaded jobs queued and running (in service) within the edge computing system.

When do we migrate a job to another edge computing node? Each edge computing node computes such threshold independently, using a Gelenbe network model (G-network). A G-network allows the estimation of the average number of jobs currently active on each edge computing node i. In the rest of the paper we denote this quantity as $E[n_i]$.

Where do we migrate the job? The destination node of the job migration mechanism is chosen according to different policies (Section IV). Each migration policy correspond to a profile. A profile of the desirable load on each node is built considering the available CPU resources and the CPU speed on each node; faster nodes and idle have priority. In particular, before selecting our destination we compute how long, on average, the hosting edge server would keep the job waiting. When we migrate we also assign a priority to avoid larger queues due to offloading requests arriving during the migration time. Note how the classically sought load balancing is only a particular case that can be achieved with our mechanisms: jobs are equally distributed among all available nodes; the offloading decision is performance-agnostic, but only consider the current node load.

To maximize the efficiency of the offloading procedure, we estimate the migration cost of each node, denoted with $c_{mig}(i, j)$. Before migrating a job from one edge computing server to another, we assess whether or not the time to solution for the offloaded task would be shorter considering the migration delay overhead.

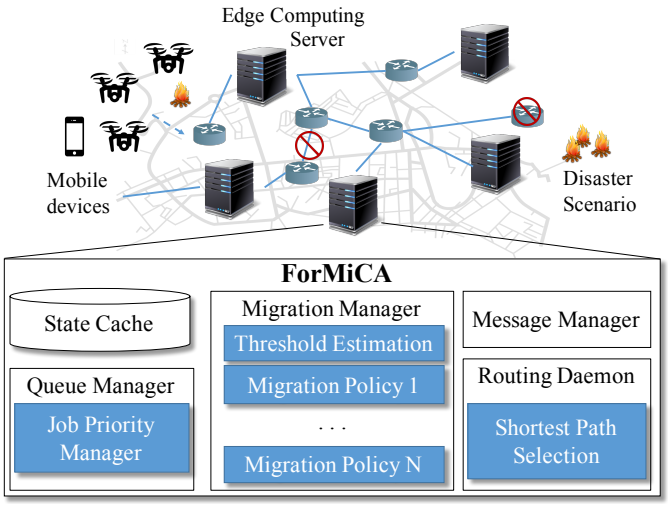


Figure 1: Cyber Foraging Overall Architecture running each edge computing server: During disasters, computationally expensive tasks are offloaded to the edge computing network from mobile devices orchestrated by first-responders; tasks are then migrated if necessary to match a latency-minimizer load profile.

Mathematically, this strategy can be formalized by the following formulation: let us assume that the considered system has K edge computing nodes; the destination node where the task will be offloaded, denoted as $dest$, is chosen by solving the following equation:

$$dest = \arg\min_{j}(d(i,j)I(i,j)), \tag{1}$$

where $I(i,j)$ is the indicator function, defined as:

$$I(i,j) = \begin{cases} 1, & \text{if } E[n_i] > \frac{\sum_{l=1}^{K} E[n_l]}{K} \wedge c_{mig}(i,j) < c_{mig}(i,i) \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Note that $0 < j < K$ and $d(i,j)$ is the known delay between server i and j.

III. Cyber Foraging Architecture and Prototype

In this section, we describe the prototype of our proposed cyber foraging architecture. We merely describe the fundamental building blocks of the architecture, while in the next sections we explore further the intelligence behind some of the most novel and crucial mechanisms.

Consider the disaster scenario in Figure 1 (top). Save and rescue operations can be facilitated by the use of new technologies. For instance, first-responders or civil smartphones, sensors and unmanned aerial vehicles (UAV) can gather imagery of missing people or of the current state of available evacuation routes, to detect the presence of chemical or biological agents, or to provide connectivity to the nearest available radio access network [9].

These devices represent the mobile clients in the considered scenario, whose architecture is exploded in Figure 1(bottom). To optimize first responder rescue operations, to save power and to speed-up their processing, such devices exploit the higher computational resources of the edge computing infrastructure by offloading image preprocessing tasks or other computationally expensive operations. When a job is offloaded to an idle edge computing node, it is executed immediately, otherwise it is queued. Such queues become longer as the availability of edge computing nodes gets scarce, given the challenged environment.

In our prototype, each node is equipped with the following management mechanisms:

State Cache. This structure has a view of the system and keeps tracks of where each task is running: on which queue of which edge computing server.

Message Manager. This prototype software component is responsible for how the stored state is exchanged among the neighbors. Moreover, this component is responsible for the job data transfer.

Migration Manager. Each node evaluates the average number of jobs in the entire system as well as the potential migration cost. These values are then used to decide if the migration should occur or not. This architecture module also selects the destination node according to the selected policy (details in Section IV).

Queue Manager. After the system decides at time t that a job has to be migrated, such job receives a priority higher than every other jobs generated by offloading requests arrived from mobile devices at time $t+1$.

Routing Daemon. This component, easily interfaced with a software-defined controller, builds up the routing table implementing a standard Dijkstra algorithm. The cost metric considers the minimum transmission delay between the two nodes.

In the next sections we expand with more details our migration managers and load profiler blocks, starting from the queuing estimator model used by the migration manager.

A. Gelenbe Network Driven Offloading

In this section we show how our migration management component of our cyber-foraging architecture leverages a G-network model to evaluate the threshold that triggers the task migration.

Intuitively, when the utilization of each queue becomes too low compared to the (estimated) average in the system, a node may request a task migration from other edge servers (only if the migration cost would justify the move). When instead the queue utilization becomes too high, or it is estimated to be too high, $i.e.$ the system is experiencing an high failure rate, our load profiling algorithm proactively redistributes the job loads.

To compute the close form of our dynamic load profiling threshold, we model the network of edge computing servers with a Gelenbe network (G-network) of queues [10]. We assume that each node has a single queue that stores all the tasks to be executed. The size of the queue is dynamic and it can shrink or enlarge according to the state of the system. We assume that a (larger) set of mobile nodes surrounding

the edge servers geo-location generates offloading requests to be queued or instantly executed. To minimize the job running time, our proposed system orchestrates the migration of jobs from busier or slower servers. We model a migration operation by moving each task to a different node's queue. Similar to previous work [11], we define a job as either a standalone task or an edge function involving multiple logically connected tasks. In case of a non-failing edge computing server, if the queue is empty, the lifetime of a job is merely modeled by the transmission time to reach the closest edge computing server from the mobile node and the job execution time. Otherwise, the lifetime of a job may be prolonged due to the waiting time that spent in the queue. In case of failing node, all jobs still to be executed are reassigned ($i.e.$, migrated). Since a task (subset of a job) may be reassigned multiple times to failing nodes, its completion time depends on the time that it has spent in all failing node queues, plus the time spent in the queue of the node that executes it (holding or service time).

Formally, we assume our system to be an *open* migration process consisting of $\mathcal{K} = \{1, 2, \ldots, K\}$ edge computing server queues. The term *open* indicates that tasks can enter or exit the system. Moreover, we assume that the service time of each edge computing server follows an exponential distributions with service rate μ_k, $1 \leq k \leq K$.

The G-network model envisions two type of customers: positive or negative. Positive customers increment the queue and receive service as ordinary queuing network customers. Therefore, they represent both offloading requests generated by mobile nodes and jobs that migrate among the queues. Negative customers instead decrease the number of positive customers in the queue and do not receive any service. If a queue is empty, negative customers do not have any effect. Therefore, they represent "control" signals that trigger the job migration.

We denote by λ_k the external arrival rate of positive customers to queue k and by r_k the external arrival rate of negative customers to queue k. Moreover, a customer which leaves queue i goes to queue j as a positive customer with probability $p^+_{i,j}$, or as a negative customer with probability $p^-_{i,j}$. It may also depart from the network with probability d_i.

A vector $\mathbf{n} = \{n_1, n_2, \ldots, n_K\}$ defines the number of jobs belonging to each of the $\mathcal{K}$ edge computing server queues. The vector $\mathbf{n}$ is assumed to be a Markov process with state space:

$$\mathcal{N} = \{\mathbf{n} : n_k \geq 0, k = 1, \ldots, K\} \tag{3}$$

At the steady state, the distribution of number of tasks in each queue, or being executed, obeys the "product form" distribution, $i.e.$, it can be written as the product of the probability function depending on the single node's queues, if the external positive or negative customer arrivals are Poisson, the service times of positive customers are exponential and independent, and if the movement of customers between queues is Markovian.

***Proposition** 3.1:* For each node k, the average arrival rate of a positive customer in its queue is given by $\Lambda^+_k = \sum_{q=1}^{K} \rho_q \mu_q p^+_{qk} + \lambda_k$ and the average arrival rate of a negative customer is given by $\Lambda^-_k = \sum_{q=1}^{K} \rho_q \mu_q p^-_{qk} + r_q$. If $(n_1, \ldots, n_m)$ is the steady state probability that there are n_k tasks in the k^{th} node's queue for $k = 1, 2, \ldots, K$, and if we assumed that each node can execute at most one job at a time, the external positive or negative customer arrivals are a Poisson process, the service times of positive customers are exponential and independent, and the movement of customers between queues is Markovian, there is a steady state distribution, then such steady state probability is:

$$\pi(n_1, n_2, \ldots, n_K) = \prod_{i=1}^{K} (1 - \rho_i) \rho_i^{n_i}. \tag{4}$$

Proof: Equation 4 is a straightforward corollary of Theorem 1 at [10] (product form of a G-network for the stationary probability distribution π). ■

***Corollary** 3.1:* We can estimate the number of tasks in each queue of an edge computing system at the steady state as follows:

$$E[n_i] = \frac{\rho_i}{1 - \rho_i} \tag{5}$$

where the utilization factor ρ_i of node i's queue is defined by:

$$\rho_i = \frac{\lambda^+_i}{\mu_i + \lambda^-_i}. \tag{6}$$

Proof: All edge computing nodes are independent elementary systems of queues, using proposition 3.1 and Little's law [12], we have the claim. ■

Our migration manager uses these close forms to evaluate the threshold that triggers (or not) the task migration. In the next section we describe the migration policies, the other crucial element of our migration manager (Figure 1).

IV. Migration Manager and Migration Policies

Our Cyber foraging architecture is policy-based. In this section we describe such policies that can be used when our migration manager triggers a job migration. Although our system is designed to support multiple policies, in our evaluation we focus on a representative set of the migration policies. The destination edge server node (future host) can be chosen based on the following criteria:
(i) **Load-balancing**. Tasks are equally distributed among all available nodes, $i.e.$, the destination is uniformly selected by the migration manager among all active and connected nodes;
(ii) **Load profiling**. This is classical: a profile of the available nodes is shaped, according to the computation resources, and the destination is chosen according to the built profile. Our prototype supports any profile $i.e.$, a node with a lower average service time has a higher probability to be selected.

To evaluate the beneficial of each policy, a migration cost, $c_{mig}(i, j)$ from node i to node j, is evaluated by the migration manager. The migration cost c_{mig} is evaluated as the sum of the following times: (i) transmission delay of the task to go

from the current node to the new one, (ii) time interval it should wait in the queue before it can be processed and (iii) a fixed time due to processing operations needed to start the migration. Formally, we have:

$$c_{mig}(i,j) = d_{trans}(i,j) + (n_j - 1)\frac{1}{\mu_j} + d_{proc} \qquad (7)$$

where n_j, $d_{trans}(i,j)$, d_{proc} are the number of jobs in the node j, the transmission delay from node i to node j through the shortest path evaluated by using the transmission delay metric, and the processing time to compute the migration cost, respectively.

(iii) **Harmonic**. As third migration policy, we use Harmonic [13], [14], a well-known randomized algorithm designed to solve the k-server problem. The k-server problem is the problem of efficiently move k servers on the nodes of a graph G according to a sequence of requests (a request is a set of k-points). The problem's aim at minimizing the total distance covered by the servers to reach the requested points. As each request arrives, an algorithm to solve the k-server problem must determine which server to move to the requested point. Our migration problem is slightly different than the k-server problem but we still use a version of the Harmonic algorithm to solve it. In particular, a migration policy that we tested chooses each edge computing server j, destination of the task migration, with probability:

$$p_j = \frac{(c_{mig}(i,j))^{-1}}{\sum_{q=1}^{N}(c_{mig}(i,j))^{-1}}. \qquad (8)$$

Why do we use Harmonic? In a disaster (or any other challenged) scenario, network connections or servers are likely to be unavailable, and offloading requests from first responders arrive in an online fashion. Our (logically centralized) migration manager has to decide where to migrate without knowing the full set of request, and the future availability of the destination edge servers. An oblivious adversary may place every failure exactly on the most powerful or idle server that would be picked as a job destination. To combat the oblivious adversary, it is known that a probabilistic approach helps: the adversary, cannot purposely fail the picked destination as it is chosen with a stochastic process.

In the next section we compare those policies showing how, counter-intuitively, using the Harmonic policy may not help. In particular, the randomization helps less and less as the resources become scarce.

V. Evaluation

To establish the practicality of our architecture, we compare the performance of its policies over an event-driven simulator, written in C++, whose code is available at [7].

All components of the architecture are tested within the event driven simulator, but the core of the architecture can be easily interfaced with an SDN controller. All network topologies are generated with BRITE [15], the Boston University topology generator.

A. Evaluation Scenario

We generate (with BRITE) two types of topologies: the dynamic network of offloader first-responder mobile devices and the static network of edge computing servers. Network topologies may follow the Barabasi-Albert (scale-free) model or the Waxman (preferential attachment) model. We only show results relative to the Barabasi-Albert model as we did not find significant dependencies from the network model. Edge computing servers are placed in a predefined geographical area and network delays are assigned to each edge infrastructure link.

Table I: Parameters setting.

Parameter	Value
Average Node Density $[node/km^2]$	10, 50, 100, 150
Mobile Nodes	10, 50, 100
Maximum generation time $[s]$	1, 2, 3
Maximum service time $[s]$	1, 2, 3
Average speed $[km/h]$	3

We consider M mobile nodes randomly placed within the disaster area. Each mobile node represent a drone, an Unmanned Aerial Vehicle (UAV) or merely a smartphone ready to offload a task to the edge network. We assume that mobile nodes move at a constant speed, v. Each mobile node m generates jobs according to an exponential distribution, with average generation rate γ_m. These jobs are uploaded to the closest edge computing server through their wireless link. When a job arrives at the selected edge computing node, it may be immediately executed or inserted in a queue. Each edge computing server k processes a job in its queue according to an exponential distribution, with average service rate μ_k. The parameter settings are shown in Table I.

In our evaluation we consider the following approaches:

- **Baseline Approach.** The baseline scenario does not involve any migration mechanism. Tasks arrive at a specific node and wait their turn in the queue until they are served by the first edge server. This is the scenario envisioned by the vast majorities of the previous work on offloading. As we will see, this scenario is often suboptimal in terms of offloaded job completion time minimization.
- **Threshold-based Approach.** When the queue size of an edge node reaches a pre-configured threshold, its jobs migrate to an other queue according to a given policy. We implemented three policies as described in Section IV: load-balancing, load-profiling policy and stochastic-based (Harmonic).
- **Migration Cost-based Approach.** Jobs move only if it is worth, that is, when the queue size of an edge server exceeds the average number of jobs in the system, tasks migrate minimizing their migration cost overhead.

We evaluate all approaches during stable network conditions (without nodes or link failures) as well as with random link failures. When a node fails, all its tasks are migrated to another node according to the policy chosen by the migration

manager. In particular, we consider three percentage of nodes failure: 10%, 50%, and 90%. 10% means that during the simulation, we let 10% of the link fail at predefined time interval.

B. Evaluation Results

To evaluate the proposed cyber foraging policy-based architecture, we compare performance of different policies estimating the time to complete 500 offloaded jobs. After the 500^{th} job is complete our simulation stops. In all our simulations we assume that a job is equivalent to a task, but an interesting open question would be to understand how to optimally offload several tasks per job.

In the rest of this section we investigate three scenarios and for each we pinpoint the main take-home messages of our evaluation. The three scenarios differ in the edge computing failure model; in the first simulation campaign, we consider a network edge without failures; in our second simulation campaign we consider the edge network with random failures; in the third we consider a more powerful adversary that forces the most powerful edge computing servers to fail, $i.e.$, it would guarantee the worst possible performance without migration. If there are two servers with equal power we pick the server with the largest amount of jobs in its queue; ties are split at random.

1) Without Failures Onloading: In this subsection we summarize the key findings when neither the edge network nor the edge computing server fail during the duration of our simulations.

- *Mobile-driven offloading shows worse performance than back-end driven onloading.* The implementation of the onloading Load Balancing (LB) policy or the onloading Load Profiling (LP) policy (with or without migration cost) leads to a lower job completion time, as shown in Fig. 2 (a) and (b). This is expected, since onloading allows the idle edge server capacity to be exploited. In particular, in case of a small network size, the gap between the mobile-driven offloading and the onloading strategies is higher. As shown in Fig. 2 (a) and (b), when the network size is 10 nodes, LB and LP significantly outperform the mobile-driven offloading.
- *Load Profiling (LB) outperforms Load Balancing (LB).* LP allows to exploit more efficiently system resources. The selection of server with lower average service time allows faster jobs completion. This result is shown in Fig. 2 (a), (b), and (c). In particular, the Cumulative Distribution Function of job completion time shows that the number of completed jobs by implementing the LP policy is always lower than the LP policy. Surprisingly, the use of migration costs does not bring significant advantages to the total reduction of the job completion time. This is an interesting finding that requires further investigation.

2) Random Failures Onloading: In this subsection, we summarize the key findings when edges network or edge computing servers fail during the duration of our simulation.

- *Onloading LP outperforms onloading LB.* Even in case of failures, LP allows to exploit more efficiently system resources. This is shown in Fig. 2 (d), (e) and (f). The higher the percentage of failed nodes, the higher the time to complete the total number of jobs. This is because, a lower number of edge servers are available, hence their queue size increases. Higher queue sizes are more likely to trigger the migration mechanism. Moreover, when the network size decreases, the job completion time increases because the number of available edge servers decreases, therefore jobs can be distributed on a lower number of nodes.
- *The lower the number of edge servers, the higher the job completing time.* When the number of servers decreases, the queue size of the available servers increases, resulting in higher job completion time.

3) With Failures of the most powerful servers Onloading: In this subsection, we summarize the key findings when the most powerful edge computing servers fail during the duration of our simulation and compare them to the stable case (without failure).

- *The Harmonic policy shows better performance than onloading LB policy with and without failure.* Powerful edge servers allow to serve jobs faster by reducing queue waiting time. This is shown In Fig. 3.

VI. Related Work

Cyber foraging [8], [4] combines resource-constrained mobile devices with powerful static servers by offloading code to surrogates for remote execution. Applications can be partitioned at different granularity levels: (i) application level, to offload virtual machines, (ii) task model level, to offload application elements that can be executed in a sequential or parallel order; (iii) and method level, to offload functions or code fragment.

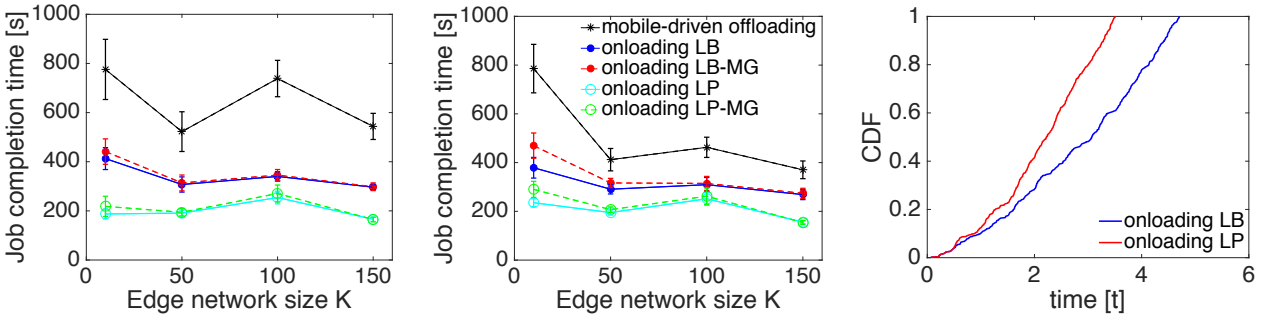
Research on offloading systems has followed two different directions [16], [6]: client-driven and back-end driven.

In client-driven offloading systems, like MAUI [17], CloneCloud [18] and Comet [19], the mobile device offloads parts of programs on a single-server to solve the mobile energy constrains. Problems of this approach are due to the heterogeneity of devices, the use of an accurate code profiling, and optimal offload conditions, that often need continuous monitoring of network conditions.

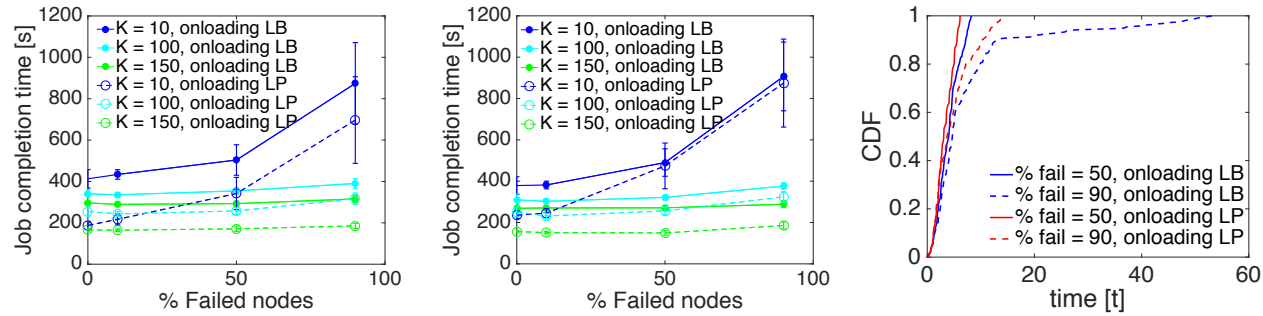
In back-end driven offloading systems, such as Cloudlet [8], the edge cloud is inherently designed to handle heterogeneous devices, and can practically access to unlimited energy and computational resources.

We propose a cyber foraging architecture that subsumes client-driven offloading and back-end driven onloading strategies by merely instantiating a few policies.

Some architectures use run-time methods to ensure that the workload is distributed to the appropriate servers when

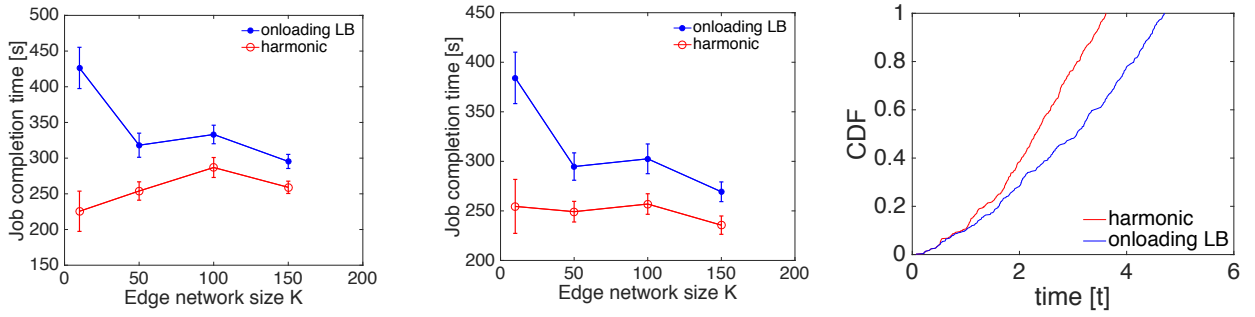


(a) Without failure, M = 50 (b) Without failure, M = 100 (c) Without failure, K = 10, M = 50

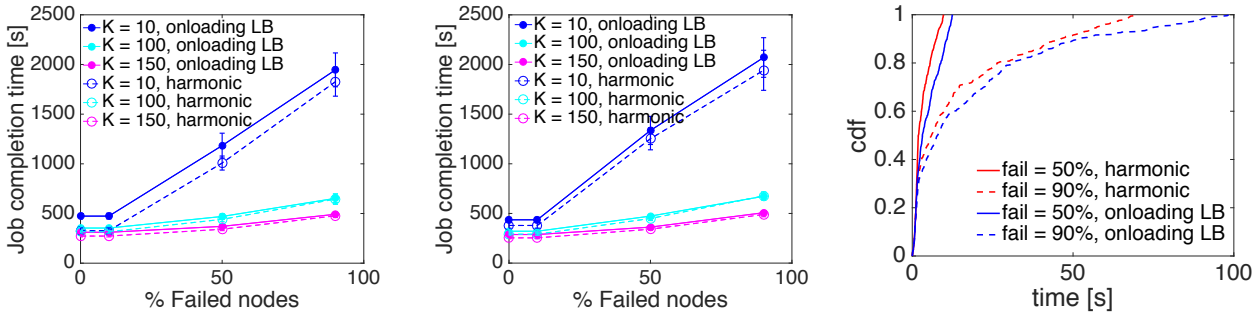


(d) With failure, M = 50 (e) With failure, M = 100 (f) With failure, K = 10, M = 50

Figure 2: Job completion time and cumulative distribution function with mobile-driven offloading, onloading LB, and onloading LP policies.



(a) Without failure, M = 50 (b) Without failure, M = 100 (c) Without failure, K = 10, M = 50



(d) With failure, M = 50 (e) With failure, M = 100 (f) With failure, K = 10, M = 50

Figure 3: Job completion time and Cumulative Distribution Function with harmonic and onloading LB policies.

the user requests arrive at the cloud in real time [20]. Differently, offline algorithms [21] aim at balancing over-utilized resources on the edge servers by assuming that all the task information is known a priori or has regular patterns.

Our architecture policies are based on the *Compare and Balance* approach [22], [23] that aims at reaching an equilibrium condition of servers and at efficiently managing workload in the system. This is achieved by selecting a node by a probabilistic basis and comparing its load to the one of the current hosts. The extra load is migrated if the selected host has a lower expected job completion time.

VII. Conclusion

Cyber foraging is a central mechanism in edge computing; resource-intensive jobs may start on mobile devices to finish within a powerful server in geographical proximity. This technology has opened the path to latency-sensitive mobile applications. In challenging (network) environments, such as those created by a natural or man-made disaster scenario, all cyber foraging challenges are exacerbated by two problems: edge nodes are scarce and hence easily overloaded and failures are common due to the ad-hoc hostile conditions.

To cope with these challenges, in this work we have proposed FORMICA, a policy-based architecture for cyber foraging and job migration orchestration. With a prototype and extensive simulations, we have shown that the cyber foraging mechanism is more efficient when it is managed by the back-end edge computing infrastructure. FORMICA proactively orchestrates offloaded jobs from mobile devices by leveraging a result in queuing theory known as the G-network product form. This result is used to estimate the number of jobs currently in each edge server. This information is then used to proactively reaches a load, defined with an high level policy profile.

Among our simulations results, we found that offloading jobs from mobile devices to the edge cloud is insufficient to minimize their completion time (and so back-end onloading is a responsive alternative to classical mobile-driven offloading). Second, in disaster scenarios or in other challenged environments, randomized job scheduling solutions may improve cyber foraging performance.

Acknowledgment

This work has been partially supported by the National Science Foundation award CNS-1647084.

References

[1] A. Ahmed and E. Ahmed, "A survey on mobile edge computing," in *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, Jan 2016, pp. 1–8.

[2] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "Mobile edge computing: Survey and research outlook," *CoRR*, vol. abs/1701.01090, 2017.

[3] J. Wang, J. Pan, and F. Esposito, "Elastic urban video surveillance system using edge computing," in *Proceedings of the Workshop on Smart Internet of Things*. ACM, 2017, p. 7.

[4] M. Sharifi, S. Kafaie, and O. Kashefi, "A survey and taxonomy of cyber foraging of mobile devices," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 4, pp. 1232–1243, 2012.

[5] K. Bhardwaj, M. W. Shih, P. Agarwal, A. Gavrilovska, T. Kim, and K. Schwan, "Fast, scalable and secure onloading of edge functions using airbox," in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, Oct 2016, pp. 14–27.

[6] F. Esposito, A. Cvetkovski, T. Dargahi, and J. Pan, "Complete edge function onloading for effective backend-driven cyber foraging," in *2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct 2017, pp. 1–8.

[7] FORMICA source code. (2017) https://github.com/agneseven/FORMICA.

[8] M. Satyanarayanan, "A brief history of cloud offload: A personal journey from odyssey through cyber foraging to cloudlets," *GetMobile: Mobile Computing and Communications*, vol. 18, no. 4, pp. 19–23, 2015.

[9] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, and K. Ha, "The role of cloudlets in hostile environments," *IEEE Pervasive Computing*, vol. 12, no. 4, pp. 40–49, Oct 2013.

[10] E. Gelenbe, "G-networks: a unifying model for neural and queueing networks," *Annals of Operations Research*, vol. 48, no. 5, pp. 433–461, Oct 1994. [Online]. Available: https://doi.org/10.1007/BF02033314

[11] W. T. Su and C. Y. Kao, "Estito: An efficient task offloading approach based on node capability estimation in a cloudlet," in *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, March 2017, pp. 1–6.

[12] K. S. Trivedi, *Probability and statistics with reliability, queuing and computer science applications*. Chichester, UK, UK: John Wiley and Sons Ltd., 2002.

[13] Y. Bartal and E. Grove, "The harmonic k-server algorithm is competitive," *J. ACM*, vol. 47, no. 1, pp. 1–15, Jan. 2000. [Online]. Available: http://doi.acm.org/10.1145/331605.331606

[14] F. Esposito and W. Cerroni, "Geomig: Online multiple vm live migration," in *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*, April 2016, pp. 48–53.

[15] A. Medina, A. Lakhina, I. Matta, and J. W. Byers, "BRITE: An Approach to Universal Topology Generation," in *MASCOTS*, 2001.

[16] K. Bhardwaj, M.-W. Shih, P. Agarwal, A. Gavrilovska, T. Kim, and K. Schwan, "Fast, scalable and secure onloading of edge functions using airbox," in *Edge Computing (SEC), IEEE/ACM Symposium on*. IEEE, 2016, pp. 14–27.

[17] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 49–62.

[18] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 301–314.

[19] M. S. Gordon, D. A. Jamshidi, S. A. Mahlke, Z. M. Mao, and X. Chen, "Comet: Code offload by migrating execution transparently." in *OSDI*, vol. 12, 2012, pp. 93–106.

[20] Q. Xia, W. Liang, and W. Xu, "Throughput maximization for online request admissions in mobile cloudlets," in *38th Annual IEEE Conference on Local Computer Networks*, Oct 2013, pp. 589–596.

[21] H. Shen, "Rial: Resource intensity aware load balancing in clouds," *IEEE Transactions on Cloud Computing*, 2017.

[22] Y. Sahu and R. Pateriya, "Cloud computing overview with load balancing techniques," *International Journal of Computer Applications*, vol. 65, no. 24, 2013.

[23] Y. Zhao and W. Huang, "Adaptive distributed load balancing algorithm based on live migration of virtual machines in cloud," in *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*. IEEE, 2009, pp. 170–175.