Simulation Modelling Practice and Theory 101 (2020) 102045

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Machine learning-Based traffic offloading in fog networks

George-Eduard Zaharia[a], Tiberiu-Alex-Irinel Şoşea[a], Radu-Ioan Ciobanu[a,*], Ciprian Dobre[b]

[a] Faculty of Automatic Control and Computers, University Politehnica of Bucharest, 313 Splaiul Independentei, Bucharest, Romania
[b] National Institute for Research and Development in Informatics, Bucharest, Romania

A R T I C L E I N F O

Keywords:
Fog
Mobile
Wi-Fi
Mobile broadband
Multi-path TCP
Offloading
Machine learning
Genetic algorithm

A B S T R A C T

Wi-Fi offloading in fog networks is believed to be one of the best ways to solve the significant data increase in cellular networks, since nodes located close by are used as relays for offloading traffic and computations. This alarming growth has affected these networks and has put its mark on their performance. Some operators might try upgrading the wide area networks, but in most scenarios, this is not the most cost-effective and optimal solution. These operators would benefit more from intelligent offloading solutions. Therefore, we can use Wi-Fi networks to send some of the packets, thus releasing cellular networks and decongesting traffic. This paper deals with offering a complete offloading solution and presents various profiles aimed at different purposes: saving the battery, getting the maximum data rate, or balancing the two, as well as offering a simulator that reproduces the behavior of the devices in an environment as close to reality as possible. Through extensive analysis, we show that the proposed solutions are able to improve certain metrics based on user requirements.

1. Introduction

By 2022, the cellular traffic per month for each smartphone will reach 25 GB in developed countries. In just six years, the traffic is going to skyrocket, increasing by as much as 1000% [1]. Due to this overwhelming growth, the cost of maintaining an acceptable performance is also rising significantly. The cost is a massive influencer in the user experience [2]. At the same time, 95% of these data will come from cellular networks, and, although static traffic is still higher than mobile traffic, the growth rate of the mobile data flow (66%) is surpassing the data flow of static devices (21%) by a large margin [3]. In consequence, instead of upgrading the WANs (wide area networks), we are looking for a more software-oriented solution, in the form of fog networks. In such networks, devices located close to the actual network nodes (and usually connectable through Wi-Fi) are employed as intermediary nodes, used for traffic and computation offloading (otherwise referred to as Wi-Fi offloading).

Data offloading is a solution that can significantly reduce the amount of data being carried on the cellular networks by sending the data via a different path, the Wi-Fi, thus freeing bandwidth for other users. The solution should take into account the degree of coupling between the cellular and Wi-Fi networks, which is high when the mobile network operator itself deploys Wi-Fi access points, and low otherwise, since there is no pre-established connection between the two kinds of networks. Intelligent offloading ideas can deliver [4], and this paper will present possible solutions for an optimal implementation. The solution will try and maximize different user properties, such as data rate or battery consumption. This is not such a trivial task, as the battery consumption highly depends on

* Corresponding author.
E-mail addresses: george.zaharia0806@stud.acs.upb.ro (G.-E. Zaharia), tiberiu_alex.sosea@stud.acs.upb.ro (T.-A.-I. Şoşea), radu.ciobanu@cs.pub.ro (R.-I. Ciobanu), ciprian.dobre@cs.pub.ro (C. Dobre).

https://doi.org/10.1016/j.simpat.2019.102045
Received 1 July 2019; Received in revised form 20 November 2019; Accepted 16 December 2019
Available online 17 December 2019

the Wi-Fi signal strength [5]. A user who is running low on battery might want to minimize the battery consumption and still be able to receive his emails, whereas someone else might want to have a smooth video streaming experience at all times, so we might want to maximize the throughput in this case. In order to be able to apply these solutions, there has to be a tool that is capable of reproducing real-world scenarios.

The motivation behind solving this problem is to significantly lower the operating costs while also improving the user experience and performance. Taking into consideration the high amount of data that is continuously being transmitted, the need to perform intelligent offloading decisions is increasing substantially every year. Wi-Fi offloading is chosen because it is already widespread and it incurs a low cost [6]. At the same time, the most recent solutions to solve data offload problems are simple and do not provide a robust and complete solution [7]. Perhaps the strongest case for data offloading, though, is the fact that it creates a faster and more reliable experience for users. Rather than deal with crowded networks and slow response times, data offloading ensures that customers get the data rates that they expect and can remain productive wherever they happen to be. In fact, with so much data being offloaded these days, it is actually becoming a key consideration for consumers when choosing a wireless provider.

However, there are some other compelling reasons that service providers and MVNOs (mobile virtual network operators) are relying more heavily on data offloading for. For starters, devices with both built-in Wi-Fi and cellular capabilities (still) work better on Wi-Fi networks [8]. This is especially true indoors, so using data offloading can help ensure reliable service no matter what device is used in any environment. Another (for some, even more compelling) reason that data offloading is gaining momentum is that it reduces costs for providers, and ultimately, for customers. For providers, this happens because they do not need additional equipment in order to satisfy quality of experience requirements. For customers, this may lead to lower overall prices for data subscriptions. Furthermore, when the data rates are very low due to many connections in crowded areas, the perceived quality of experience by the users is extremely low, which affects the operator negatively, making it incur losses in the long run. Offloading allows providers the flexibility to add bandwidth and capacity in areas where it is most needed, such as in major metro areas or on university campuses, without a great deal of time and expense. This improves customer experience and reduces churn.

Thus, the main contribution of this paper is to propose and evaluate three approaches for Wi-Fi offloading, based on a greedy algorithm, multi-criteria optimization, and a genetic algorithm. To achieve this goal, we first build a simulator based on machine learning to predict both Wi-Fi and mobile broadband signal strengths, in order to determine the battery consumption and expected data rate. Through experimental evaluation, we show that our proposed solutions are able to improve data rates and battery consumption in various scenarios.

The rest of this paper is structured as follows. Section 2 presents related work in the area of offloading in general and Wi-Fi offloading in particular. Then, Section 3 presents data collected in the Politehnica University of Bucharest area and the way the collection was performed. Using this data, we propose and present a static and mobile node simulator in Section 4. Section 5 proposes several solutions for data offloading in networks composed of static and mobile devices, and then in Section 6 we evaluate them. Finally, we draw conclusions and present future work in Section 7.

2. Related work

Mobile data offloading is the use of auxiliary network technologies aimed at boosting the capacities for mobile data delivery. One can use it whenever the signal is unsatisfactory, or there is data congestion on the channel. There is a need for mobile data offloading because of the most recent surge in data traffic [1]. The leading complementary network technologies used for mobile data offloading are Wi-Fi, femtocell, and Integrated Mobile Broadcast.

More and more devices feature Wi-Fi capabilities, and Wi-Fi access is becoming more widely available in homes, enterprises, and retail locations. Wi-Fi offloading is emerging as an attractive option for network operators [7]. Femtocell is suitable for lower throughput than Wi-Fi [9]. This example [10] also has integrated LTE and Wi-Fi networks. Small cells transmit on both licensed and unlicensed bands. Finally, Integrated Mobile Broadcast (IMB) is part of the Rel. 8 standard. It enables spectrally-efficient broadcast data delivery using Time Division Duplexing (TDD) radio techniques [11].

We will be focusing on Wi-Fi offloading by reviewing the leading heuristics [9] and provide a complete classification of the most current solutions. Mobile data offloading techniques can be classified depending on the assumptions one can make on the level of synergy between cellular and unlicensed wireless networks.

Firstly, we should take into account the requirements of the app generating the traffic we are analyzing. One requirement would be the time the app is willing to trade for other metrics (such as battery). This incurs the addition of a temporal dimension. This way, we could say app traffic is ϵ-delayable, i.e., it can suffer ϵ seconds of delay. For the rest of the paper, these two categories will be called non-delayed offloading and delayed offloading.

2.1. Non-delayed offloading

In non-delayed offloading, every transmission offers a hard performance delay constraint defined by the application. Except for the packet processing and physical transmission, no delay should be added. Audio and video streaming, for example, work in real-time and are unable to hold extra delay. Voice transmission has a tolerable latency of 50ms. A network should be able to meet this requirement consistently, which is why opportunistic networks are unfeasible for voice transmissions. On the other hand, transparent handover and interoperability between the alternative access technologies and the operator hardware should be the primary focus (currently, the above are not met when offloading through IEEE 802.11 access points). Voice Over IP (VoIP) offloading would be offered a nearly transparent offloading process this way.

One way to reduce congestion in mobile networks is to deploy well-placed static access points, motivating the significant recent interest in data offloading. We can now compute the amount of data that can be offloaded with this configuration. A way to improve data offloading would be to find an optimal AP placement so that we can maximize the amount of data being transmitted on the alternate routes.

In terms of the standardization efforts made by 3GPP (3rd-Generation Partnership Project), the LTE network designed an Evolved Packet Core (EPC) [12], i.e., access-independent all-IP based architecture to meet hybrid network requirements. The EPC delivers IP-based services with access technologies such as mobile broadband and Wi-Fi. Both the 3GPP technologies, as well as non-3GPP technologies [13], are supported. Data offloading is a critical offloading option to overcome the overcrowded network problem. The solution proposed is ANDSF (Access Network Discovery and Selection Function [14]), which triggers the change between different access technologies, also taking advantage of the EPC hybrid architecture.

Multi-interface integration is a new IP-based transport protocol that is a much-needed feature for boosting offloading capabilities. Offloading does many switchovers, so multiple access technologies aggregation is a must. This system is designed to balance the throughput and cost, maximizing the cost-utility metric.

2.2. Delayed offloading overview

Delayed offloading is a technique to prolong a packet transmission for a more suitable time in the future. There are two main types of delayed offloading traffic. The first kind is traffic with a quality of service (QoS) limit, which is a relaxation of the non-delayed offloading. Although the packet has to be transmitted intact, there can be certain relaxations like deadlines and certain QoS metrics that have to be met. On the other hand, fully delay-tolerant offloading refers to a relaxation of the delivery constraint. This is a considerable aspect, as we can now move traffic opportunistically. A lot of mobile applications do not require real-time performance, so losing the real-time support (because of the delays implied) is not such a big deal for specific applications such as downloading app updates, synchronizations, etc. At the same time, this means that we get an alternate solution for data manipulation during peak hours, enabling us to add delays to improve specific metrics like battery and data rate Delay-tolerant apps, on the other hand also rely on geographical position, as we could delay the traffic until a user reaches an IEEE 802.11 access point (AP) or a neighbor that carries the content of interest, increasing the offloading capacity.

In general, Wi-Fi efficiency shades mobile broadband performance, so prediction-based offloading deals with exploring the energy efficiency of a delayed transmission. We make a decision based on predictions regarding future transmission performance, representing it as a function of the RSSI (Received Signal Strength Indicator). There have also been conducted synthetic and real-world experiments to confirm the excellent performance of SALSA [15], which can save up to 40% of energy if compared to other baseline offloading strategies. The core of the system relies on DTP (Delay Tolerant Protocol) to get smooth experience and separate it from the interferences of the application layer.

A substantial improvement of the offloaded traffic ratios remains delay-tolerance growth. The average data transmission time for a delayed offloading is usually lower than the deadline. It is also possible that, with specific packets, delaying the transmission might lead to higher average data rates, so better completion times. This is mainly due to the usually high data rates of Wi-Fi compared to mobile networks.

2.3. Machine learning in traffic offloading

In recent years, the idea of employing machine learning (ML) in traffic and computation offloading (and in networking in general) has caught on, with many works addressing this research topic. Some of the earliest work focused on applying ML in wireless sensor networks in order to solve problems such as energy-aware communication, efficient sensor deployment, resource allocation, or task scheduling [16], while more recent uses of ML in networking include traffic prediction and classification, routing strategies, quality of experience optimization, congestion control, or performance prediction [17].

Nowadays, the utility of machine learning methods in network offloading tends to focus on moving code from a mobile node to other devices, which can be mobile, but also edge, fog, or cloud nodes [18,19]. For example, Shi et al. propose a method for computation offloading where the access to resources is intermittent and of variable quality [20]. The authors attempt to predict node connectivity and execution time for various tasks through ML methods on Android devices. Other ML methods for offloading computations include fuzzy-sets and evidence-based learning [21], instance-based learning and naive Bayes [22], decision trees and perceptron-based learning [23], or deep learning [24].

The method proposed in this paper does not deal only with computation offloading, but also focuses on traffic offloading, where requests and replies are sent and received with the help of other nodes in the network. Furthermore, we propose three separate ML-based approaches for offloading, namely a greedy approach, multi-criteria optimization, and a genetic solution.

2.4. Key aspects for implementation of Wi-Fi offload

Mobile network operators should have the opportunity to make agreements with Wi-Fi ISPs (Internet Service Providers) to enable their infrastructure to be suitable for traffic offloading. This should increase the Wi-Fi network ubiquity, thus making it mobile data traffic offloading-friendly.

Minimal to no user interaction is needed to initiate Wi-Fi offloading, and seamlessness offloading is essential for the user experience. An intelligent connection manager mechanism at the user equipment end should be provided to produce flexibility and efficiency. This mechanism should choose the proper network, amount of offloaded data, as well as application or service-specific offload.

There is no universal authentication mechanism for both Wi-Fi and mobile networks, since each technology has its tools and authentication process. On the other hand, some techniques like SIM-based Wi-Fi authentication are becoming more and more popular. At the same time, there have only been proprietary implementations of policy and charging rules in offloading situations. ANDSF (Access Network Discovery and Selection Function) and PCRF (Policy and Charging Rules Function) servers developed by 3GPP are trying to inflict standards for policy and charging rules in these types of situations.

3. Data collection

To ensure the accuracy of the results of the offloading algorithm, and also to differentiate the various decision criteria for choosing a particular interface, it was necessary to collect data. Data was collected through a series of experiments, both on static devices and on mobile devices. This data was then fed to the simulator presented in Section 4 and, based on the results obtained, we were able to develop the solutions proposed in Section 5. The simulator output is a machine learning model designed to predict as close to reality as possible real-world metrics and values. This means that the quality and quantity of the collected data are directly proportional to the precision of the model.

3.1. *Collecting the data on static devices*

The data collecting process was sustained on devices called MONROE nodes. Using various interfaces (Wi-Fi, mobile broadband - MBB, or MultiPath TCP - MPTCP), the data was collected for longer periods and then interpreted. Below we will describe the way we collected the data, and we will also provide information about the MONROE nodes.

3.1.1. *MONROE Nodes*

The MONROE platform [25,26] represents an element of fundamental importance for developing and improving MBB networks. The initial development of the MONROE platform took place within the EU Horizon 2020 project MONROE[1]. The MONROE platform offers an environment for a broad range of experiments. As far as we are concerned, we used it for networking experiments, testing interface data rates and collecting the results.

The MONROE nodes are based on PC Engines APU2D4 motherboards, having three mini-PCIe slots, one of them supporting a 3G/4G modem, an accelerated processing unit (APU) with a 1-GHz 64-bit dual core AMD Geode, 16 GB SSD and 4 GB RAM. At the same time, an external USB hub with per-port-power-switching (PPPS), a Compex WLE600VX Wi-Fi adapter and three ZTE MF910 LTE CAT4MiFis are connected. The node supports the 802.11n, 802.11g, 802.11b, 802.11ac, and 802.11a Wi-Fi standards, with a frequency range of 2.412-2.472 GHz and 5.180-5.825 GHz.

3.1.2. *Node placement and setup*

To vary the conditions and to test different circumstances, we placed two MONROE nodes (called node A and node B) in our faculty campus, in different places. The nodes were in two separate local networks, and were accessible from the outside through two gateways, as shown in Fig. 1. For the Wi-Fi network interface of a MONROE node, we connected it to a pre-determined access point, which was the one with the best signal in each of the two nodes' locations. In order to collect data about MBB behavior, we also used different types of SIM cards for mobile broadband, namely an Orange M2M SIM and an Orange PrePay SIM. For the MPTCP data collection experiments, we excluded the Ethernet interface and only directed traffic through MBB and Wi-Fi.

3.1.3. *Running the experiments*

A good way to measure an interface data rate is to download files from a remote server. To accomplish this, we set up an MPTCP-capable machine located in the same local network which hosts several files with different sizes (from 5MB to 200MB). Then, back on the nodes, we used *wget* to download the files. The experiments were repeated for different interfaces so that they have the data rates associated with them. Besides Wi-Fi and MBB, we tested with MPTCP, including the three ways to schedule it: *default, roundrobin*, and *redundant.*

To cover a wide range of possibilities, we had to run the experiments during different times of the day. For efficiency, we used a scheduler that, at certain time intervals, calls a script which executes a file download and saves the output After a few attempts, we found that the best interval is 30 minutes, since it captures variations in data rates (when we tested with one-hour intervals, almost 30% of the variations seen with the smaller interval were lost; on the other hand, testing with 15-minute intervals only yielded about 4% more drastic variations). The *wget* results are then parsed, and the output is saved in a file that contains the location from where the download was initiated, the location which stored the files, as well as the timestamp. For each file size, we saved the time it took to download it and the data rate. These files can then be fed to the simulator, such that we will observe the data rate fluctuations, as well as which interface to choose depending on the time of the day. A sample of the collected data can be found in Fig. 2, which shows that, in this particular scenario, the data rates are more or less constant regardless of the file size, and that Wi-Fi clearly outperforms MBB. This is the case for our static nodes, but we need to analyze if this holds true for the mobile nodes as well. There is also the

[1] https://www.monroe-project.eu/themonroeproject/

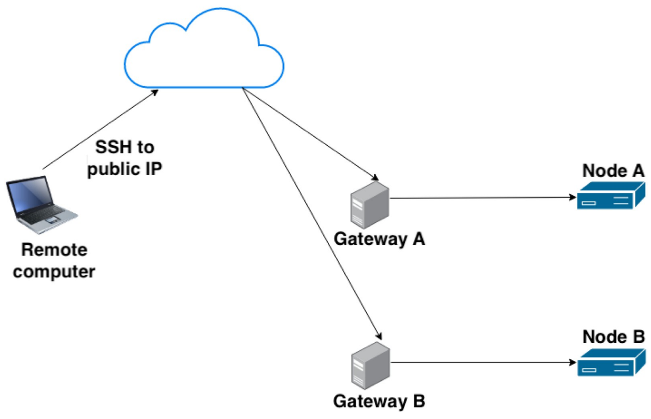


Fig. 1. Connecting to the MONROE nodes.

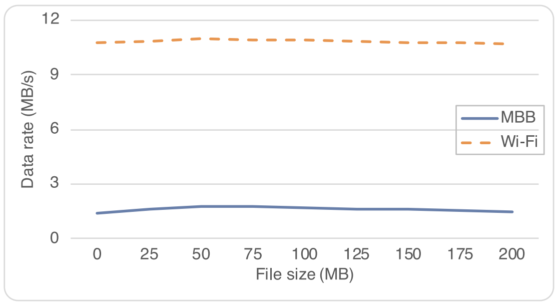


Fig. 2. Sample data collected from a static node.

question of energy consumption, especially important in fog networks where nodes can be battery-powered IoT devices. Node A has a -72 dBm signal strength. We want to see what happens when the signal strength weakens. Thus, we must broaden our signal strength spectrum and we chose to use mobile nodes, so we can compute the Wi-Fi performance in a much wider space.

3.2. *Collecting the data on mobile devices*

Considering that, this time, the devices are mobile, we need to adapt and collect data as we move. For this, we implemented and employed an Android application (called Wi-Fi Scanner[2]) which records access point data such as location, SSID, capabilities, and the timestamp the access point has been encountered. We gathered data from a group of 15 users in the area of our faculty for several months, ending up with information for more than 120 access points, thus making it possible to create a map of the Wi-Fi networks in the campus. The collected data is then fed into the simulator described in Section 4. For simulating the MBB connections of these mobile devices, we use the same data from the static devices, since they use the same type of SIM cards.

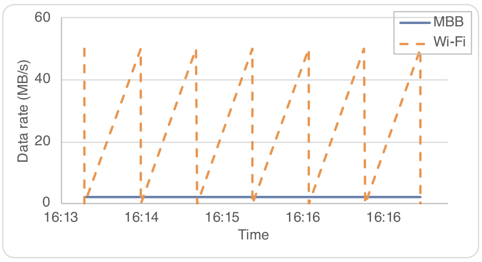
3.3. *Data preprocessing*

Once all this data is collected, we have to convert it to a more machine-learning friendly form, so we chose a.csv format. There will also be another data preprocessing step inside the simulator that will merge the two datasets, after learning how to predict the data from the static devices. This comes as a necessity because the static nodes are not able to “scan” as many Wi-Fi networks as mobile devices, so being able to predict MBB performance at any time is needed.

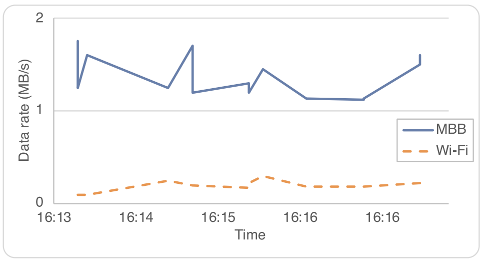
3.4. *Data analysis*

We take a look at how the collected data is structured, assuming that every Wi-Fi network has a 50 Mb/s data rate. The data is continuous over a week (there is no data shortage in a specific day, e.g., Sundays) and this is very important, as the values collected show that the MBB or Wi-Fi performance (at fixed latitude and longitude) tend to have minimal fluctuations in the same days of different weeks (e.g., Sundays at latitude X and longitude Y there is always high data usage). A example of a visualization of our data when two Wi-Fi access points are located at different floors can be found in Fig. 3, where it can be observed that there are high

[2] https://github.com/raduciobanu/WiFiScanner/



(a) Access point at the sixth floor



(b) Access point at the seventh floor

Fig. 3. Sample data collected for three minutes by a mobile node regarding two access points located at different floors of the same building.

fluctuations in data rates for the closer access point. We can already get some insights regarding how an offloading algorithm aiming to maximize the data rate might work on this data. While it would always choose MBB in a scenario such as the one shown in Fig. 3, in Fig. 3 it might choose Wi-Fi to offload some of the traffic.

4. Traffic offloading simulator

We built the simulator because we concluded that we must know the device functioning scenario even in situations that we did not cover throughout the data collection experiments. The Python-based simulator allows us to track the evolution of our devices at any time and provides valuable information about the performance that they can achieve.

To be able to cover all possibilities, we need techniques that allow us to get results according to several input parameters. Therefore, we have found that the use of machine learning notions such as support vector machines (SVM) is a reliable solution [27]. Given a training dataset, support vector machines allow us to classify and predict values for data that is not present in the training set. By using different kernel types, we can improve the performance of the classifier, thus delivering results that are very close to reality. However, in order for the predictor to work precisely, the dataset must cover a wide range of values to cover unexpected cases. Running the simulator involves the following steps: data collection and parsing; modifying the data so that it can be fed as an input to the SVM; using the dataset to train the SVM; obtaining the model; predicting the values for the desired ranges; plotting the device evolution.

The simulator offers two options; we can choose between static nodes and mobile devices. Thus, it is almost necessary to separate the two components in two different classes: one for mobile, the other one for static. Depending on our choice, the approaches taken by the simulator are slightly different. We experimented with different types of estimators [27] to maximize accuracy and get as close as possible to the real results: SVC (C-Support Vector Classification), SVR (Epsilon-Support Vector Regression), Random Forest Classifier [28], Ada Boost Classifier [29], and Ada Boost Regressor [29].

4.1. Static node prediction

Knowing that the static nodes are not affected by as many parameters (such as location, movement speed, battery etc.) as mobile devices, there is no need for an SVM to simulate their behavior. We can feed the collected data to the simulator and, depending on the timestamp, it will predict the data rate value for a particular interface.

We first have to convert the Unix Time into the week day and the time in that specific day. Then we round that to half an hour, and add the current data rate to the mean. As trivial as this may seem, it is highly accurate (99.5%), as shown in Fig. 4.

4.2. Mobile node prediction

Once the simulator can accurately predict the MBB data rate (with a user inputted accuracy), a final dataset is created using the

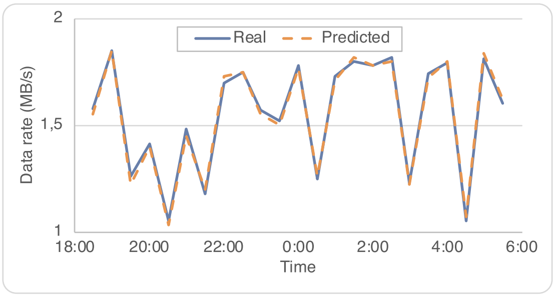


Fig. 4. Example of static node prediction for a twelve-hour interval on 27–28 March 2019.

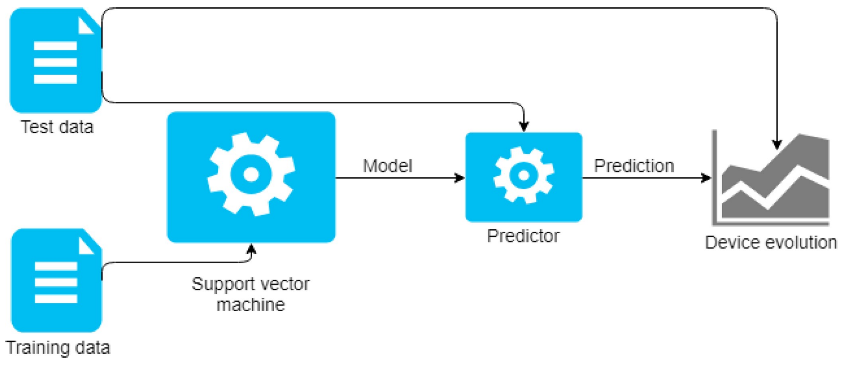


Fig. 5. Simulator workflow.

```
1: procedure DetermineEnergyConsumption
2:     if $signal\_strength \geq -60$ and $signal\_strength \leq 0$ then
3:         return -0.01 * $signal\_strength$ + 6.5
4:     if $signal\_strength \geq -70$ and $signal\_strength < -60$ then
5:         return -0.02 * $signal\_strength$ + 5.9
6:     if $signal\_strength \geq -80$ and $signal\_strength < -70$ then
7:         return -0.03 * $signal\_strength$ + 5.2
8:     if $signal\_strength \geq -85$ and $signal\_strength < -80$ then
9:         return -0.58 * $signal\_strength$ - 39.3
10:    if $signal\_strength \geq -90$ and $signal\_strength < -85$ then
11:        return -3.2 * $signal\_strength$ - 262
```

Algorithm 1. Determining the energy consumption.

static node predictor. While the primary variable in static node prediction was only the time (bound in the Monday-Sunday interval), now the input space is much larger: latitude, longitude, and time. We thus use a couple of machine learning models to try and predict the Wi-Fi signal strength.

The output of the simulator is intended to be a function which represents the evolution of the device based on the data that the user provides. The output shows information for each timestamp the device passes. Fig. 5 presents the steps taken in order to obtain the final result. The mobile nodes accuracy is highly dependent on the static nodes because MBB predictions are made only with the static node data. Because of the rather trivial and repetitive MBB performance, MBB predictions are not forecasted through a machine learning model, but rather as a mean.

For determining the signal strength in different circumstances, we need to use ML notions such as SVMs, estimators and predictors. Python offers a library for Machine Learning, represented by *scikit-learn*. It can be used for multiple purposes: from preprocessing to clustering, regression and classification. It contains a series of classifiers (Random Forest, AdaBoost) and regressors (AdaBoost) that we use for training and predicting the outcome.

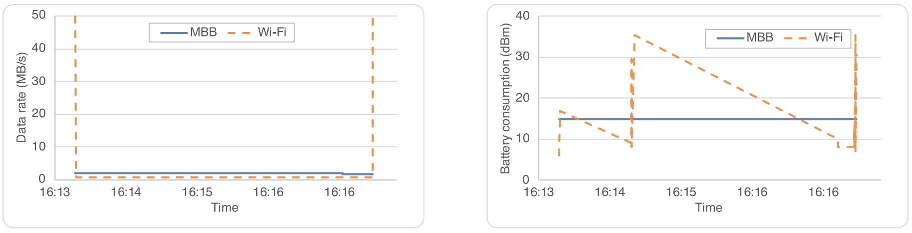
After predicting the signal strength, we can easily compute the battery consumption, as proposed in [5]. We can split the signal strengths in several intervals, depending on the energy consumption, and we can use this pattern to create a function that finds the desired values. As Ding et al. state [5], these intervals have their respective ends for a series of reasons. For low signal strengths (below -80dBm), the idle energy increases following the growth of the average idle interval between frames. At the same time, if the signal strength is found in the [-80dBm, -50dBm] range, the Power Saving Mode (PSM) tail energy and the idle energy directly affect the levels of energy drain.

Based on this information, we can develop the function we will use to compute the value of energy consumption. We consider that the values located between the interval ends can be obtained by applying a linear function which receives as input the signal strength in decibel-milliwatts (dBm) and outputs the energy consumption in micro Ampere hour (mAh). The procedure is presented in Algorithm 1.

Another problem we encountered is represented by computing the Wi-Fi data rate from the signal strength. This was done by using the information from Celtrio[3], such that we elaborated an algorithm that receives the signal strength and the bandwidth as input and outputs the Wi-Fi data rate.

The estimator choice represents an essential part of the training phase. It directly influences the performance and the entire outcome of the simulator. There are several options to choose from. The AdaBoost classifier and regressor have their performance enhanced by the machine learning meta-algorithm AdaBoost [30]. The training takes place faster and the results are generally better. We can vary parameters such as the learning rate or the number of estimators in order to experiment with different solutions. The Support Vector Regression (SVR) works with continuous values and estimates the outcome accordingly; its purpose is to set a certain threshold for the error. Once the estimator is chosen, we can proceed to train the model. If we intend to save time later, we can write

[3] http://www.celtrio.com/support/documentation/coverazone/2.0.1/basics.sensitivity.html



(a) Predicted data rate (b) Predicted battery consumption

Fig. 6. Example of data predicted in the simulator for a given access point. The variation at the start and end of the data rate sample is caused by the mobile node moving farther away and then back closer to the access points. The variation in predicted battery consumption is given by the change in signal strength during the sample interval.

the model to a file and load it afterward. Then, the model is used to predict the values. The results are then plotted, for a better understanding of the evolution.

In our experiments, the best competitor turned out to be the AdaBoost regressor, with an accuracy of 98% (with an allowed error of 1). Now that the signal strength is predicted with high accuracy, we can take a look at the battery consumption and data rate visualization. Fig. 6 shows an example of predicted battery consumption and data rate in our proposed simulator for a given Wi-Fi access point.

4.3. *Drawbacks of the simulator and the solutions in general*

First of all, the simulator learns about the different metrics and the performance a device should have in certain parts of the university campus. In other words, it predicts the hardware of the mobile network operator or the wireless. So what happens when the mobile operator upgrades the hardware, or when we install new Wi-Fi networks? The solution would ignore these changes unless new data is fed into the simulator to force the predictor to learn these new metrics. Ignoring them would result in massive performance drawbacks, so new scans should be conducted periodically to sense these changes and then to further train and sharpen the predictor.

In highly dynamic networks (and especially in mobile ones), these changes in network state need to be addressed more thoroughly. One solution for this problem would be to employ stream processing instead of classic prediction algorithms [31], which implies continuously querying the data and detecting conditions on the fly, within a small time period from actually receiving the data. However, such a solution would be extremely computationally-intensive, so alternative methods of performing these computations in due time without largely affecting a device's battery consumption are required. One example was proposed by Yang et al. [32] and it assumes that a mobile cloud can be composed of multiple co-located mobile devices, which act as providers for various computations. A collaboration between such nodes can lead not only to more efficient data stream processing, but also to a better view of the network environment, since multiple nodes perform separate observations that can be aggregated together in a crowd computing-based manner. We do not address stream processing in this paper, but we aim to extend our solution with streaming capabilities in future work.

Another advantage brought forth by the mobile cloud computing paradigm (which assumes that mobile nodes can act as both providers and clients in a cloud-like framework) is that it can alleviate some of the computing load placed upon individual mobile nodes. All the machine learning algorithms proposed here need to either run on the mobile devices themselves or in a cloud backend, if and when the mobile nodes have access to one, so the impact can be high. However, access to a backend or even to a more complex three-layered infrastructure such as the previously-proposed Drop Computing [33] can offer additional benefits, and we wish to extend our solutions to work in such an environment in the future. Another interesting discussion here is the topic of battery consumption caused by processing the ML solution on the devices. In [34], we analyze the benefits of employing Drop Computing for computation offloading, which would be a suitable approach for reducing the ML computation load.

5. Proposed solution

The offloading solutions are meant to maximize specific metrics or balance them. For example, a particular user might want to maximize their data rate or minimize battery consumption. In some instances, we can achieve a much better battery consumption with almost identical transfer data rate. We can do this by package delaying - the simulator might predict that, in some time, the signal strength is going to be much better and, as a consequence, so will the battery consumption and data rate. On the other hand, we cannot delay every packet transmission, such as packets of an application that highly depends on real-time data (e-mails, video streaming), but delaying, for example, an application update might be a good idea. Our proposed solutions tackle both delayable and non-delayable data offloading solutions. Basically, depending on the way of usage, there are different performance options: some users want to get maximum data rate regardless of the power consumption, others want to save battery and download/upload files whose presence is not urgent. At the same time, some users want shared performance, maximizing the data rate for minimal battery consumption.

```
for all entry $e$ in $dataset_N$ do
    if $e.id$ is even then
        if $e.wifi.battery < e.mbb.battery$ then
            $L_B \leftarrow L_B \cup e.wifi$
        else
            $L_B \leftarrow L_B \cup e.mbb$
    else
        if $e.wifi.datarate > e.mbb.datarate$ then
            $L_R \leftarrow L_R \cup e.wifi$
        else
            $L_R \leftarrow L_R \cup e.mbb$

$avg_B = compute\_avg(L_B)$
$avg_R = compute\_avg(L_R)$

for all entry $e$ in $dataset_{N+1}$ do
    $max_R = \max(e.wifi.datarate, e.mbb.datarate)$
    $min_B = \min(e.wifi.battery, e.mbb.battery)$
    if $max_R > avg_R$ and $min_B < avg_B$ then
        if $max_R - avg_R > avg_B - min_B$ then
            choose iface with highest data rate
        else
            choose iface with lowest battery usage
    else if $max_R > avg_R$ then
        choose iface with highest data rate
    else if $min_B < avg_B$ then
        choose iface with lowest battery usage
    else
        choose random iface
```

Algorithm 2. Multi-Criteria Optimization.

We can fulfill all these requirements by choosing the right labels for the dataset entries, depending on what we are following, since we predict signal strength and deduce energy consumption through the simulator, as shown in Section 4. We can split the signal strengths in several intervals, depending on the energy consumption, and we can use this pattern to create a function that finds the desired values.

We now can build a new dataset that we will use for training our predictor. The dataset contains four features, represented by the Wi-Fi data rate and the Wi-Fi battery consumption (depending on the signal strength), both of them determined by using the procedures described in Section 4, as well as the MBB data rate and the MBB battery consumption, learned from the static nodes experiments. The label choice will dictate which requirement we will fulfill: prioritizing the data rate, the battery, or getting the best possible results by balancing the two metrics.

5.1. Greedy approach

To maximize one metric (data rate or battery gain), we take a greedy approach. Thus, if we want to prioritize the data rate, we compare the data rate values for Wi-Fi and MBB. If the Wi-Fi value is higher than the MBB one, we set the label to 0, otherwise we set it to 1. At the same time, for maximum energy consumption efficiency, we compare the battery consumption values. If the Wi-Fi value is lower than the MBB value, we, similarly, will set the label to 0, otherwise we set it to 1.

5.2. Multi-criteria optimization

Another strategy, which maximizes gain for battery and data rate, is shown in Algorithm 2, where $dataset_N$ contains the first N values from the dataset, while $dataset_{N+1}$ contains the rest of the dataset. The proposed solution stores some N best values in two different lists, one for data rate (L_R) and one for the battery (L_B). For the first N entries in the dataset, we alternate: we choose the best value for each metric, depending on the row's parity (as shown at lines 1–11 from Algorithm 2). After the first N rows, we now have two lists, and we can compute the mean value for them. For row $N + 1$, we check the percentage gain for data rate and battery. If the best data rate for that entry is higher than the average data rate and the best energy consumption is lower than the average battery consumption, we choose the one which has better growth, as shown at lines 19–23 of Algorithm 2. Otherwise, we want the one which respects our condition. If neither data rate nor battery respects the condition, we randomly choose one metric. After that, we pop the first element from the list corresponding to the metric and insert the new value.

Similarly, we can experiment with a new strategy, one that checks the local gain. We can compare the data rate difference between the two interfaces, as well as the battery difference, and we can turn them into a percentage. We want to maximize this percentage, so we choose the metric which offers us the best gain.

We can then feed the dataset, composed of the four attributes columns and the label column, to a predictor, to train it. Similarly to the simulator, we can choose between different types of estimators. The output is represented by a model, which we can store in order to save computation time. The model is used for predicting the interface, so it outputs either 0 for Wi-Fi, or 1 for MBB.

On the other hand, knowing that we continuously supply the static nodes with energy, our primary purpose is to maximize their

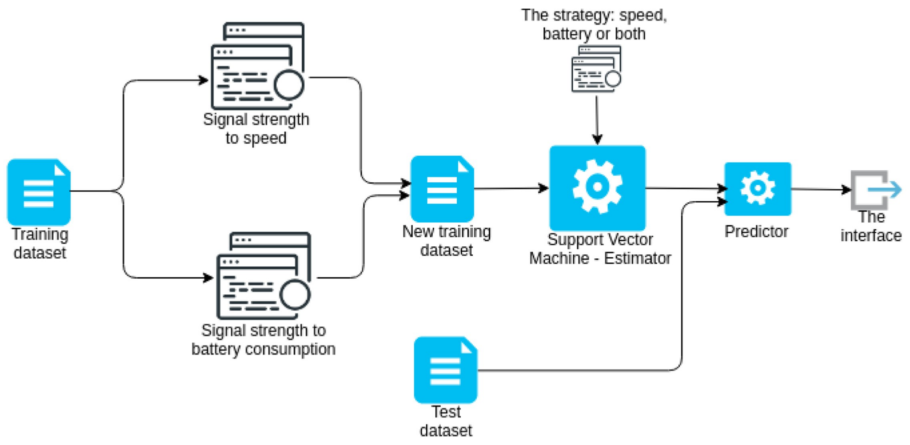

The strategy: speed, battery or both
Signal strength to speed
Training dataset
New training dataset
Support Vector Machine - Estimator
Predictor
The interface
Signal strength to battery consumption
Test dataset

Fig. 7. Interface prediction workflow.

download and upload data rates. We even have more interface options, besides the Wi-Fi and MBB. We can use the MultiPath TCP with its scheduling options, *default, roundrobin*, and *redundant*. We collect the data obtained from the static experiments, we parse it, and then we build a dictionary, which stores the interface values for a specific time slot. The strategy is to choose the best data rate value for the current timestamp, based on this dictionary. The algorithm outputs the best interface to choose, in terms of data rate.

All of the above solutions prove to be useful for the problem they want to solve. More details will be provided in the Section 6. Fig. 7 presents the workflow described above. The training set contains the data we gathered and which we will use to improve the prediction performance. The more entries (and hence more covered cases), the higher the performance. At the same time, the test set contains the new situations that the simulated device will encounter. The predictions must be as close to reality as possible.

5.3. Genetic algorithm-based solution

Although the simple feature optimization is straightforward and trivial, when multiple metrics come into the equation there have to be tradeoffs between the different features. How much can a transfer be delayed? How much better (in terms of other metrics) does a packet transmission delay D_1 has to be to outperform a packet transmission with delay D_2? Is there a way we can boost non-delayable transmissions?

Although it might not seem obvious at first, this problem fits genetic algorithms (GAs) very well. First of all, we have to find a way to compute the gain, a simple way to represent it being as $\frac{data\_rate}{battery\_consumption}$. If we define a packet through its size X (in MB) and its accepted delay Y (in hours), we need to devise a solution that firstly rejects impossible queries, and then attempts to send the package in less than Y hours.

5.3.1. Individual representation

An individual represents all the offloading decisions made in the query time interval. We divide the interval into equal segments and take individual decisions on each of these segments, as shown below:

$$\begin{pmatrix} Gene_1 & Gene_2 & Gene_3 & \cdots & Gene_n \\ Decision_1 & Decision_2 & Decision_3 & \cdots & Decision_n \end{pmatrix} \tag{1}$$

We then define fhe following parameters: dcs_i is the offloading decision of gene i, $y_j(dcs_i)$ is the decision of gene i and individual j, $gene\_no(ind_i)$ represents the number of genes of individual i, $interval\_size$ is the length of each interval (gene) in seconds, p_size specifies the size of the packet to be transmitted in MB, $b\_cons(dcs_i)$ is the battery consumed in decision i, $data\_rate(dcs_i)$ represents the average data rate in $\frac{MB}{s}$, $gain(dcs_i)$ is computed as $\frac{data\_rate(dcs_i)}{b\_cons(dcs_i)}$, while finally $traffic(dcs_i)$ is $data\_rate(dcs_i) \times interval\_size$.

Now we can get an insight into how the fitness function looks like for an individual for any purpose. We can easily choose whatever fitness function maximizes the needed metric. For example, if the goal is to minimize the battery consumption, the fitness function would look as follows:

$$fitness(ind_n) = \begin{cases} \frac{1}{\sum_{i=1}^{int\_no} b\_cons(dcs_i, int_i)} & \sum_{i=1}^{int\_no} traff(dcs_i) = p\_size \\ 0 & otherwise \end{cases}$$

Attempting to maximize the data rate would lead to the following fitness function:

$$fitness(ind_n) = \begin{cases} \sum_{i=1}^{int\_no} data\_rate(dcs_i, int_i) & \sum_{i=1}^{int\_no} traff(dcs_i) = =p\_size \\ 0 & otherwise \end{cases}$$

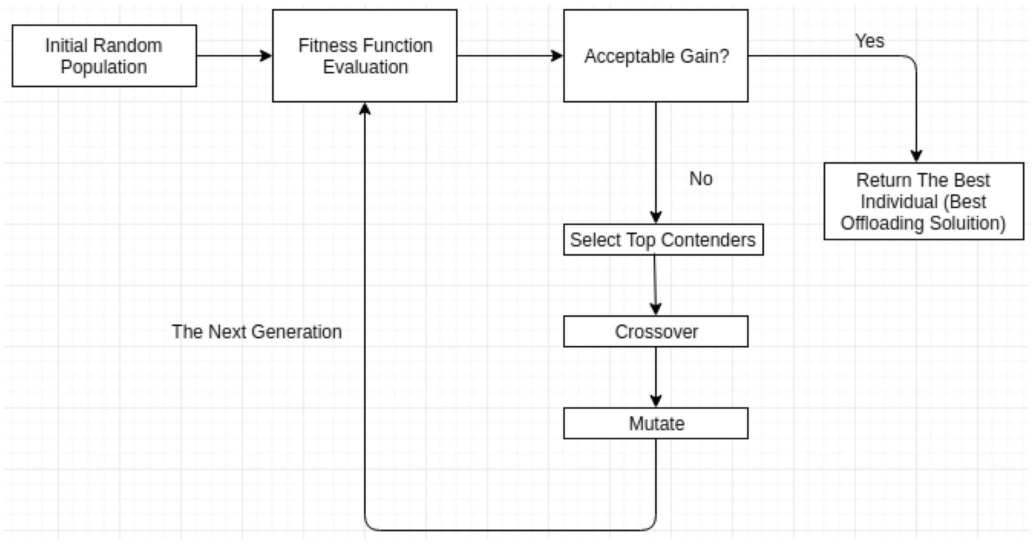

Initial Random Population
Fitness Function Evaluation
Acceptable Gain?
Yes
No
Return The Best Individual (Best Offloading Solution)
Select Top Contenders
Crossover
Mutate
The Next Generation

Fig. 8. Genetic algorithm flow.

Finally, in order to maximize the overall gain, we propose the function below:

$$fitness(ind_n) = \begin{cases} \sum_{i=1}^{int\_no} gain(dcs_i, int_i) & \sum_{i=1}^{int\_no} traff(dcs_i) = =p\_size \\ 0 & otherwise \end{cases}$$

The general genetic algorithms flow [35] can be seen in Fig. 8. In the following section, we will break down the flow and discuss about each component.

5.3.2. *Initial population*

In our proposed solution, the genes might have one of three values: *delay, Wi-Fi*, or *MBB*, which means that a 5-gene individual might have the following form:

$$\begin{pmatrix} Gene_1 & Gene_2 & Gene_3 & Gene_4 & Gene_5 \\ Delay & Wi-Fi & MBB & MBB & Delay \end{pmatrix} \tag{2}$$

We could initialize a gene randomly with one of the three values, but this might generate a lot of zero-fitness individuals (meaning that the package is not sent ultimately). It might also lead to too much data being sent (exceeding the packet length) and thus, more battery being consumed. Thus, we will randomly create individuals which try to accomplish two goals: $\sum_{i=1}^{int\_no} traff(dcs_i) \geq p\_size$ and $\forall\ i,\ such\ that\ dcs_i \neq delay,\ if\ dcs_i \leftarrow delay \Rightarrow traff(dcs_i) < p\_size$.

5.3.3. *Fitness function evaluation*

The fitness function is used to compute how fit an individual is for our scope. Depending on what we want to maximize, we use three different functions: one for battery consumption minimization, another one for data rate maximization, and a global gain maximization function for both metrics.

5.3.4. *Termination conditions*

We implemented two stop conditions, namely gain threshold and epoch limit. Both are user-inputted metrics that meet a user's requirements. For example, a person might want a fast solution, so a low epoch threshold combined with a medium gain might be suitable.

5.3.5. *Selection*

We tackled three selection techniques, starting with a simple *fitness amount-based solution*. For this one, we first compute the fitness of each individual *fitness*(*ind_i*). Then, the probability of choosing individual *I* is computed as:

$$P_i = \frac{fitness(ind_i)}{\sum_{j=1}^{ind\_no} fitness(ind_j)} \tag{3}$$

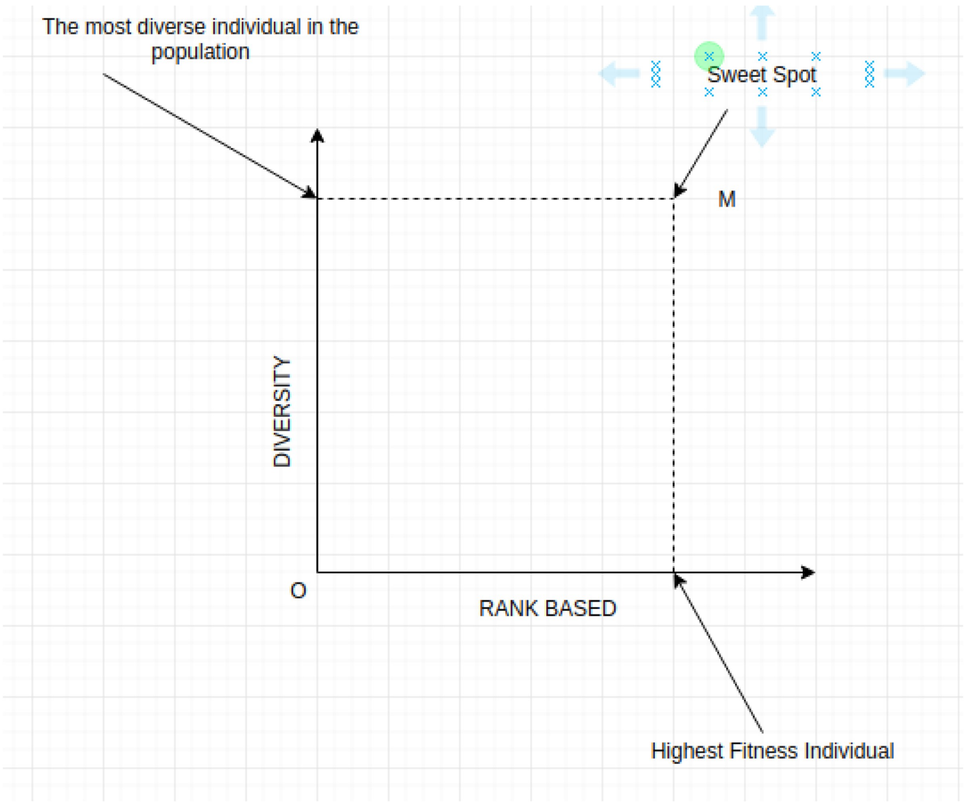

The most diverse individual in the population
Sweet Spot
M
DIVERSITY
O
RANK BASED
Highest Fitness Individual

Fig. 9. Diversity for selecting new individuals.

For the second *rank-based* solution, we choose a constant probability of P_c and compute the individual probability P_i based on this constant. We compute the fitness of each individual $fitness(ind_i)$ and then sort them in descending order. The position in the array is the rank of each individual. The first individual (the fittest) should have the highest probability. The computation is presented below:

$$\begin{aligned}&P_1 = P_c\\&P_2 = (1 - P_c)^*P_c\\&\vdots\\&P_{ind\_no} = (1 - P_c)^{ind\_no-1*}P_c\end{aligned} \tag{4}$$

The final solution is *rank and diversity-based* and uses a two-dimensional space (Rank and Diversity) to compute the probabilities. We compute the diversity as a pair-wise Hamming distance. Although the computation of the measure is quadratic, this is not going to impact the performance as much because our population is not going to exceed 100 individuals. The total Hamming distance in a population with a number of P L-gene individuals is computed as:

$$gene\_diverse(y_j(dcs_k), y_i(dcs_k)) = \begin{cases}0 & if\ y_j(dcs_k) = y_i(dcs_k)\\ 1 & otherwise\end{cases} H(pop) = \sum_{i=1}^{P-1}\sum_{j=1}^{P}\sum_{k=1}^{L}(gene\_diverse(y_j(dcs_k), y_i(dcs_k))) \tag{5}$$

The P_i probability is still computed the rank-based way. Once we have the two metrics, they can be graphically represented as in Fig. 9, which shows that we need to maximize the distance to the M point in order to create both diverse and fit individuals.

5.3.6. *Crossover*

We use crossover to diversify the individuals. The base motivation is that combining two fit individuals, there is a high probability that the resulted individual is fitter than its parents. We employ both a single point crossover and a crossover mask to bring diversity. The single point crossover combines two individuals by randomly choosing a bit, then exchanging the genes completely starting with that bit. The example below shows a crossover example by choosing bit 2.

$$\begin{array}{c}\text{Individual1:}\\ \begin{pmatrix} Gene_1 & Gene_2 & Gene_3 & Gene_4 & Gene_5 \\ Delay & Wi-Fi & MBB & MBB & Delay \end{pmatrix}\\ \text{Individual2:}\\ \begin{pmatrix} Gene_1 & Gene_2 & Gene_3 & Gene_4 & Gene_5 \\ MBB & MBB & Wi-Fi & MBB & Wi-Fi \end{pmatrix}\\ \text{Crossover1:}\\ \begin{pmatrix} Gene_1 & Gene_2 & Gene_3 & Gene_4 & Gene_5 \\ Delay & Wi-Fi & Wi-Fi & MBB & Wi-Fi \end{pmatrix}\\ \text{Crossover2:}\\ \begin{pmatrix} Gene_1 & Gene_2 & Gene_3 & Gene_4 & Gene_5 \\ MBB & MBB & MBB & MBB & Delay \end{pmatrix}\end{array} \tag{6}$$

The idea of the crossover mask stays the same, but using it provides a higher crossover result domain.

5.3.7. *Mutation*

As shown by selection, diversity is a core metric that a GA should have. Diversification is provided mainly by the mutation. The mutation further explores the solution domain. The proposed mutation is very simple, namely randomly choose a gene and then randomly change its value (i.e., offloading solution).

5.3.8. *Overview*

The Rank and Diversity approach, combined with the multi-point crossover and mutation is the way to go as it overcomes the local maximum problem found in the other two selections. This is visually represented int Section 6.

6. Experimental results

In this section, we analyze the performance of our simulator, and then of our three categories of offloading solutions: greedy, multi-objective optimization, and genetic algorithm.

6.1. *Simulator evaluation*

Data collection is very critical. A lack of data in certain locations leads to incorrect predictions in those areas. Running machine learning algorithms on incorrect data is irrelevant, so before designing solutions, we have to have a steady and reliable simulator. Thus, in order to evaluate the performance of the simulator, we started from a complete dataset that contains all the features (from the timestamp to the MBB battery consumption). We then split this dataset into two parts, one for training, one for testing. We varied the percentage that the test dataset represents from the complete dataset. At the same time, we experimented with different estimators, and we finally chose the best option for maximum accuracy. We also inserted an allowed error coefficient, relaxing the signal strength value approximation.

The experiments took place with percentage values in [0.1%, 1%, 10%], the allowed error was set to 1, and the estimators we tested were SVC, SVR, Random Forest, AdaBoost Classifier, and AdaBoost Regressor. The results are shown in Fig. 10, and we can observe that the AdaBoost Regressor offers the best accuracy and performance, most likely because, as an ensemble learning algorithm, it is able to combine multiple weak learners into a stronger one through sequential prediction.

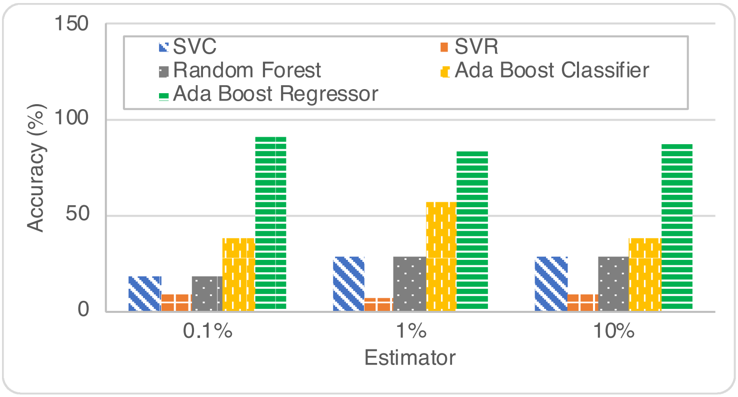
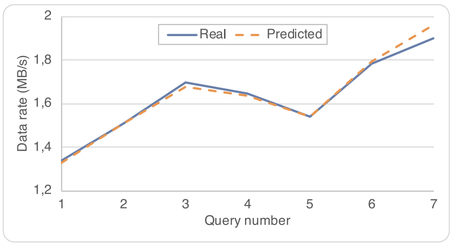
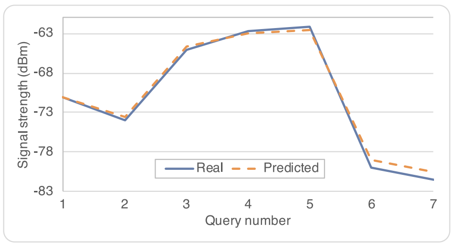


Fig. 10. Simulator prediction accuracy.



(a) Static nodes



(b) Mobile nodes

Fig. 11. Simulator prediction accuracy for static and mobile nodes.

6.1.1. *Static nodes prediction*

We also tested the accuracy of static and mobile node prediction individually. In terms of static nodes, the simulator predicts the MBB data rate, and we measured its accuracy using the precision percent (PP) metric, computed as:

$$PP = \frac{\sum_{i=1}^{test\_no} |y\_pred_i - y\_test_i| \;\; \geq \; 1 \, ? \; 0 : \; 1}{test\_no} \tag{7}$$

The overall PP obtained (for the entire static nodes dataset) was 99%, and Fig. 11a shows the prediction of the simulator for seven MBB queries.

6.1.2. *Mobile nodes prediction*

For mobile nodes, the simulator attempts to predict Wi-Fi signal strength, which then can be used to extract download data rate and battery consumption. In this case as well, the predictor precision is very favorable, with a value of 98%. As an example, Fig. 11 shows the prediction of the simulator for seven Wi-Fi queries, and it can be observed that the predicted values are very close to the real values.

Now that we have shown that the predictor is sharp, we can easily simulate different algorithms in this virtual environment. However, one should keep in mind that this performance is highly dependent on hardware and mobile network changes. Further data should be collected to keep the predictor accurate.

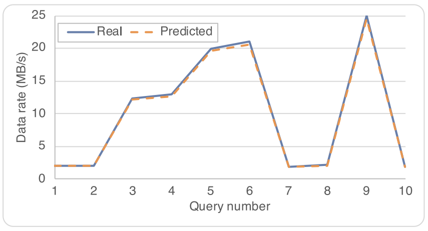
6.2. *Greedy approach performance*

The greedy predictor is designed to learn the offloading solution for single feature maximization. Fig. 12 is a visualization of single feature predictions for both metrics. The overall prediction precision in this case is 99% for data rate as the feature, and 98.5% for battery consumption.

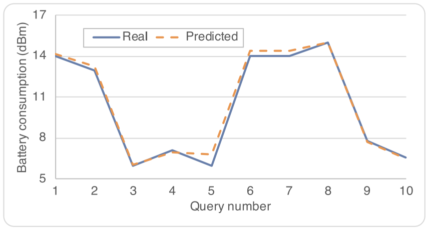
6.3. *Multi-feature optimization evaluation*

In this scenario, we evaluated our solution separately for both kind of nodes. To evaluate the gain obtained by introducing the offloading algorithm for static nodes, we ran four experiments. For the first one, only mobile data is used, without connecting to a Wi-Fi access point. For the second and third experiments, only Wi-Fi and then only MPTCP were employed. Finally, we use our algorithm to predict the best interface for maximizing the data rate (which is the goal in this scenario, since static nodes are not constrained by battery.

The experiments ran for 12 hours. When the algorithm was used, the interface was tested every 30 minutes and potentially changed to a different one. We compared the average data rate for all four scenarios, and extracted our conclusions. As Fig. 13 shows, we obtained a 1.526% data rate increase compared to MPTCP when we used the offloading algorithm. At the same time, the data rate increased with 13.29% if we only relied on Wi-Fi. The obtained data rates are the following: 1.5897 MB/s for MBB, 3.5986 MB/s for



(a) Data rate



(b) Battery consumption

Fig. 12. Greedy approach prediction accuracy.

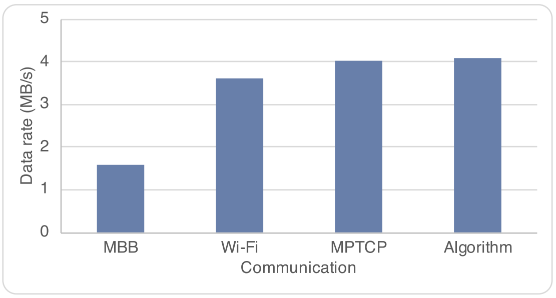


Fig. 13. Experiment results for the static nodes.

Wi-Fi, 4.0157 MB/s for MPTCP, and 4.0770 MB/s for our proposed solution.

We also had to test the performance of mobile devices. In order to do that, we considered several different scenarios implying the usage of the mobile offloading algorithms. Firstly, we considered that the device uses, in turns, the Wi-Fi and the MBB interfaces and we recorded the performances. Then we ran the three versions of our algorithm, prioritizing the data rate, the battery, as well as trying to get the best results for the two metrics. After that, we compared the results and we drew the conclusions. As we can see in Table 1 and Fig. 14, our algorithms proved to be effective. The strategy which maximizes both metrics obtained the best results and is recommended for continuous usage. The data rate is only 5.2% lower than the best possible results, while the battery consumption is 7.2% higher. On the other hand, if we intend to download a file as fast as possible, the algorithm will ensure that the traffic is offloaded to Wi-Fi networks whenever the achievable data rate is higher than the data rates attained by using mobile data at that timestamp. Furthermore, if the user's priority is battery conservation, that strategy will be switched such that the battery consumption will reach the lowest possible value. Of course, there exists a trade-off: the data rates will not be as good but, as we can see in Table 1, they will still be higher than in the scenario when we are not using the algorithm.

Thus, the conclusion here is that our algorithm helps us get improved performances in specific areas, depending on the strategy we choose.

6.4. Genetic algorithm performance

Fig. 15 is the visual representation of the gain per epoch inside the genetic algorithm. We can see that the diversity approach has a much higher slope, meaning that choosing differently and diversifying even more using mutation enables the algorithm to explore a lot more than the usual selections. Furthermore, while in the first two selections, the gain caps at 80–85, selection three caps at 160. That is probably because starting from an epoch, the mutation cannot produce new individuals, since the individuals produced are immediately eliminated because of lower fitness. This does not happen if the population has a high diversity.

In conclusion, Diversity and Rank selection combined with mutation and crossover produce proper offloading scheduling solutions. This solution will be further explored and analyzed in future work.

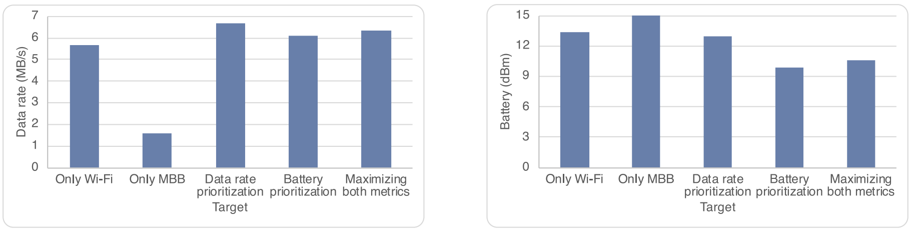
7. Conclusions and future work

In this paper, we addressed the problem of Wi-Fi offloading in fog networks, as we designed a three-component application:

- a network simulator - eases the offloading solution performance analysis; we achieved good results that enabled us to simulate against a vast amount of data (impossible in a real-world environment)
- single and multi-criteria offloading solutions - we proposed solutions for either maximizing a single or multiple parameters (such as data rate or battery) when offloading
- genetic algorithm-based offloading solution - proposed an offloading solution for both non-delayed and delayed transmissions, which was proven to have a good performance at the cost of some computation power; it is designed mainly for delay-tolerant data, although the architecture makes it possible to be applied on delay-intolerant data.

Table 1
Mobile device performance.

	Data rate (MB/s)	Battery (dBm)
Only Wi-Fi	5.6673	13.4098
Only MBB	1.5663	15.0
Data rate prioritization	**6.6679**	13.0102
Battery prioritization	6.1067	**9.8567**
Maximizing both metrics	*6.3184*	*10.5672*



(a) Data rate (b) Battery consumption

Fig. 14. Experiment results for the mobile nodes.

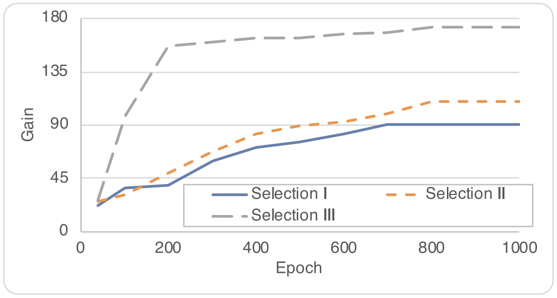


Fig. 15. Genetic offloading algorithm performance.

A genetic algorithm-based solution seems to be a suitable solution for offloading decision-making. The careful combination of a proper selection function, mutation, and crossover, as well as a careful population initialization, provide exciting results. These results can be further used to build a decision tree learning model to predict offloading decisions without the heavy, more cumbersome GA computation. The downside of the simulator and the offloading solutions is that they assume that no access points or Wi-Fi networks are added. For every equipment change, there has to be done a new data collection which implies, in turn, additional cost, which is something that we aim to address in the future.

Acknowledgments

The research presented in this paper is supported by project “vINCI: Clinically-validated INtegrated Support for Assistive Care and Lifestyle Improvement: the Human Link” (EU AAL 2017).

References

[1] Ericsson, Future mobile data usage and traffic growth, 2019, (https://www.ericsson.com/en/mobility-report/future-mobile-data-usage-and-traffic-growth).
[2] V. Roto, R. Geisler, A. Kaikkonen, A. Popescu, E. Vartiainen, Data traffic costs and mobile browsing user experience, Proceedings of the 4th MobEA Workshop on Empowering the Mobile Web, (2006), pp. 1–6.
[3] S. Eido, A. Gravey, How much lte traffic can be offloaded? Meeting of the European Network of Universities and Companies in Information and Communication Engineering, Springer, 2014, pp. 48–58.
[4] K. Lee, J. Lee, Y. Yi, I. Rhee, S. Chong, Mobile data offloading: how much can wifi deliver? IEEE/ACM Trans. Netw. (ToN) 21 (2) (2013) 536–550.
[5] N. Ding, D. Wagner, X. Chen, A. Pathak, Y.C. Hu, A. Rice, Characterizing and modeling the impact of wireless signal strength on smartphone battery drain, ACM SIGMETRICS Perform. Eval. Rev. 41 (1) (2013) 29–40.
[6] F. Mehmeti, T. Spyropoulos, Performance analysis of mobile data offloading in heterogeneous networks, IEEE Trans. Mobile Comput. 16 (2) (2016) 482–497.
[7] Qualcomm, A 3g/lte wi-fi offload framework: Connectivity engine (CNE) to manage inter-system radio connections and applications, 2011, (https://www.qualcomm.com/media/documents/files/3g-lte-wifi-offload-framework.pdf).
[8] I. Fogg, The State of Wifi vs. Mobile Network Experience as 5G Arrives, Technical Report, OpenSignal, 2018.
[9] F. Rebecchi, M.D. De Amorim, V. Conan, A. Passarella, R. Bruno, M. Conti, Data offloading techniques in cellular networks: a survey, IEEE Commun. Surv. Tutor. 17 (2) (2014) 580–603.
[10] M. Simsek, M. Bennis, M. Debbah, A. Czylwik, Rethinking offload: how to intelligently combine wifi and small cells? Proceedings of the IEEE International Conference on Communications (ICC), IEEE, 2013, pp. 5204–5208.
[11] R. Esmailzadeh, M. Nakagawa, E.A. Sourour, Time-division duplex CDMA communications, IEEE Pers. Commun. 4 (2) (1997) 51–56.
[12] J. Kempf, B. Johansson, S. Pettersson, H. Lüning, T. Nilsson, Moving the mobile evolved packet core to the cloud, Proceedings of the IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), IEEE, 2012, pp. 784–791.
[13] T.-T. Tran, Y. Shin, O.-S. Shin, Overview of enabling technologies for 3GPP lte-advanced, EURASIP J. Wirel. Commun. Netw. 2012 (1) (2012) 54, https://doi.org/10.1186/1687-1499-2012-54.
[14] S.-N. Yang, C.-H. Ke, Y.-B. Lin, C.-H. Gan, Mobility management through access network discovery and selection function for load balancing and power saving in software-defined networking environment, EURASIP J. Wirel. Commun. Netw. 2016 (1) (2016) 204, https://doi.org/10.1186/s13638-016-0707-0.
[15] N. Ioannou, K. Kourtis, I. Koltsidas, Elevating commodity storage with the salsa host translation layer, Proceedings of the IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE, 2018, pp. 277–292.
[16] Ma Di, Er Meng Joo, A survey of machine learning in wireless sensor netoworks from networking and application perspectives, Proceedings of the 6th International Conference on Information, Communications Signal Processing, (2007), pp. 1–5, https://doi.org/10.1109/ICICS.2007.4449882.

[17] M. Wang, Y. Cui, X. Wang, S. Xiao, J. Jiang, Machine learning for networking: workflow, advances and opportunities, IEEE Netw. 32 (2) (2018) 92–99, https://doi.org/10.1109/MNET.2017.1700200.

[18] J. Hauswald, T. Manville, Q. Zheng, R. Dreslinski, C. Chakrabarti, T. Mudge, A hybrid approach to offloading mobile image classification, Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), (2014), pp. 8375–8379, https://doi.org/10.1109/ICASSP.2014.6855235.

[19] S. Deshmukh, R. Shah, Computation offloading frameworks in mobile cloud computing : a survey, Proceedings of the IEEE International Conference on Current Trends in Advanced Computing (ICCTAC), (2016), pp. 1–5, https://doi.org/10.1109/ICCTAC.2016.7567332.

[20] C. Shi, P. Pandurangan, K. Ni, J. Yang, M. Ammar, M. Naik, E. Zegura, IC-Cloud: Computation offloading to an intermittently-connected cloud, Technical Report, Georgia Institute of Technology, 2013.

[21] H. Flores, S. Srirama, Adaptive code offloading for mobile cloud applications: exploiting fuzzy sets and evidence-based learning, Proceeding of the Fourth ACM Workshop on Mobile Cloud Computing and Services, MCS '13, ACM, New York, NY, USA, 2013, pp. 9–16, https://doi.org/10.1145/2497306.2482984.

[22] H. Eom, R. Figueiredo, H. Cai, Y. Zhang, G. Huang, Malmos: Machine learning-based mobile offloading scheduler with online training, Proceedings of the 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, (2015), pp. 51–60, https://doi.org/10.1109/MobileCloud.2015.19.

[23] H. Eom, P.S. Juste, R. Figueiredo, O. Tickoo, R. Illikkal, R. Iyer, Machine learning-based runtime scheduler for mobile offloading framework, Proceedings of the IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 17–25, https://doi.org/10.1109/UCC.2013.21.

[24] S. Yu, X. Wang, R. Langar, Computation offloading for mobile edge computing: a deep learning approach, Proceeding of the IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), IEEE, 2017, pp. 1–6.

[25] Ö. Alay, A. Lutu, R. García, M. Peón-Quirós, V. Mancuso, T. Hirsch, T. Dely, J. Werme, K. Evensen, A. Hansen, S. Alfredsson, J. Karlsson, A. Brunstrom, A.S. Khatouni, M. Mellia, M.A. Marsan, R. Monno, H. Lonsethagen, Measuring and assessing mobile broadband networks with monroe, Proceedings of the IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), IEEE, 2016, pp. 1–3.

[26] Ö. Alay, A. Lutu, M. Peón-Quirós, V. Mancuso, T. Hirsch, K. Evensen, A. Hansen, S. Alfredsson, J. Karlsson, A. Brunstrom, A. Safari Khatouni, M. Mellia, M. Ajmone Marsan, Experience: an open platform for experimentation with commercial mobile broadband networks, Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking, ACM, 2017, pp. 70–78.

[27] C.-C. Chang, C.-J. Lin, Libsvm: a library for support vector machines, ACM Trans. Intell. Syst. Technology (TIST) 2 (3) (2011) 27.

[28] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32, https://doi.org/10.1023/A:1010933404324.

[29] N. Duffy, D. Helmbold, Boosting methods for regression, Mach. Learn. 47 (2) (2002) 153–200, https://doi.org/10.1023/A:1013685603443.

[30] R.E. Schapire, Explaining adaboost, Empirical Inference, Springer, 2013, pp. 37–52.

[31] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. etintemel, Y. Xing, S.B. Zdonik, Scalable distributed stream processing, Proceeding of the CIDR, (2003).

[32] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, A. Chan, A framework for partitioning and execution of data stream applications in mobile cloud computing, ACM SIGMETRICS Perform. Eval. Rev. 40 (4) (2013) 23–32.

[33] R.-I. Ciobanu, C. Negru, F. Pop, C. Dobre, C.X. Mavromoustakis, G. Mastorakis, Drop computing: ad-hoc dynamic collaborative computing, Future Gener. Comput. Syst. 92 (2019) 889–899.

[34] R.-I. Ciobanu, C. Dobre, M. Bălănescu, G. Suciu, Data and task offloading in collaborative mobile fog-based networks, IEEE Access 7 (2019) 104405–104422.

[35] A. Thengade, R. Dondal, Genetic algorithm survey paper, Proceeding of the IJCA Processing National Conference on Recent Trends in Computing, NCRTC 5 (2012).