Comparative Analysis of Network Libraries for Offloading Efficiency in Mobile Cloud Environment

Farhan Sufyan[1], Amit Banerjee[2]
Department of Computer Science,
South Asian University, New Delhi, India - 110021

***Abstract*—In the modern era, smartphones are increasingly becoming an integral and essential part of our daily life. Although the hardware capabilities of the smartphones (i.e., processing, memory, battery, and communication) are improving every day, however, it is not enough to handle computation-intensive applications, such as image processing, data analytics, and encryption. To overcome these limitations, mobile cloud computing (MCC) is introduced, which augments the capabilities of smartphones and resources of the cloud to provide better QoS performance to the user. The idea is to save resources in the smartphones by offloading the computationally intensive tasks to the cloud. In this context, researchers have proposed several offloading frameworks, mainly addressing challenges of *why-what-when* and *where* to offload. In this paper, however, we explore another challenging issue of offloading, i.e., *how-to-offload*. More specifically, we analyze different networking libraries *(HttpURLConnection, OkHttp, Volley, Retrofit)* and study their performance on various dynamic factors such as data size, communication media, hardware and software of the smartphone. Our objective is to explore if an application can use the same networking library for all the smartphones and all purposes or there is a need to make an adaptive decision based on the local constraints. To understand this, we perform a comprehensive analysis of the networking libraries on different Andriod smartphones in the real environment and found that there is a need of adaptive network library selection because libraries perform changes in different scenarios.**

Keywords*—Android; Mobile Cloud Computing (MCC); network libraries; offloading; performance

I. INTRODUCTION

Over the last decade, we have seen unprecedented and exponential growth in the popularity of smartphones and smart devices. With increasing capability of the smart devices, consumers are becoming more demanding, and the developers are building more sophisticated applications with interesting features and complexity [1]. In spite of significant progress, smartphones are unable to accommodate user/application demands, particularly for applications that require resource-intensive processing, memory, and power. To solve the above problem, the concept of *computation offloading* or simply *offloading* is introduced in mobile cloud computing (MCC). Offloading is an idea that has been around for a long time and evolved from various paradigms of distributed computing. The concept gained more attention with the popularity of smart mobile devices and the demand for incorporating more sophisticated applications on these well-connected devices.

Offloading augments the capabilities of smartphones and the resources of the cloud to complement the requirements of computation-intensive applications. The computational resources in the cloud can be provisioned on-demand to augment more capabilities to smartphones. Smartphones can offload large computations or computation intensive modules to the cloud via its wireless communication network for execution and retrieve the results [2]. The primary objective of offloading is to reduce the task execution time and energy consumption of smartphones. Offloading is also referred as *cyber foraging* [3] or *remote execution* [4]. The challenges of offloading includes the following decision problems: *why, what, when,* and *where* to offload [5], Fig. 1. Researchers have proposed several offloading frameworks and techniques to address theses challenges. Broadly, an offloading decision is made by utilizing the local information of the smartphones, namely, CPU and memory utilization, code profiling, network speed and/or user behavior. These parameters are fed into an optimization engine to take optimal decision to achieve the offloading objectives [6], [7], [8], [9]. The optimization decision can be either be taken on mobile device [10], cloud [9] or both [11]. After the offloading decision, smartphone can send heavy computation [12] or network intensive tasks [13] to the cloud using a network library, as shown in Fig. 2.

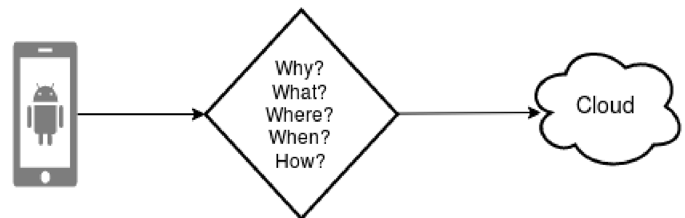


Fig. 1. Offloading Perspectives

In this paper, we investigate another challenging issue related to offloading, i.e., *how-to-offload*, particularly deals with the decision of selecting network library for transferring the computation or data from the smart devices to the cloud and vice-versa [14], [15]. Recently, the *how-to-offload* problem is also discussed in [16], where the author emphasis requirement of network library selection for realizing the offloading potentials. More specifically, we intend to investigate, if an application can use a particular network library in all smartphones or needs to select it dynamically, as shown in Fig. 3. There are several network libraries available for exchanging data in smartphones, such as HttpURLConnection, OkHttp, Volley, and Retrofit. However, selecting a particular library for an application is not straightforward, as it depends upon various dynamic factors, including file size, communication media, battery consumption, hardware and software of the smartphone. In this paper, we aim to study various network

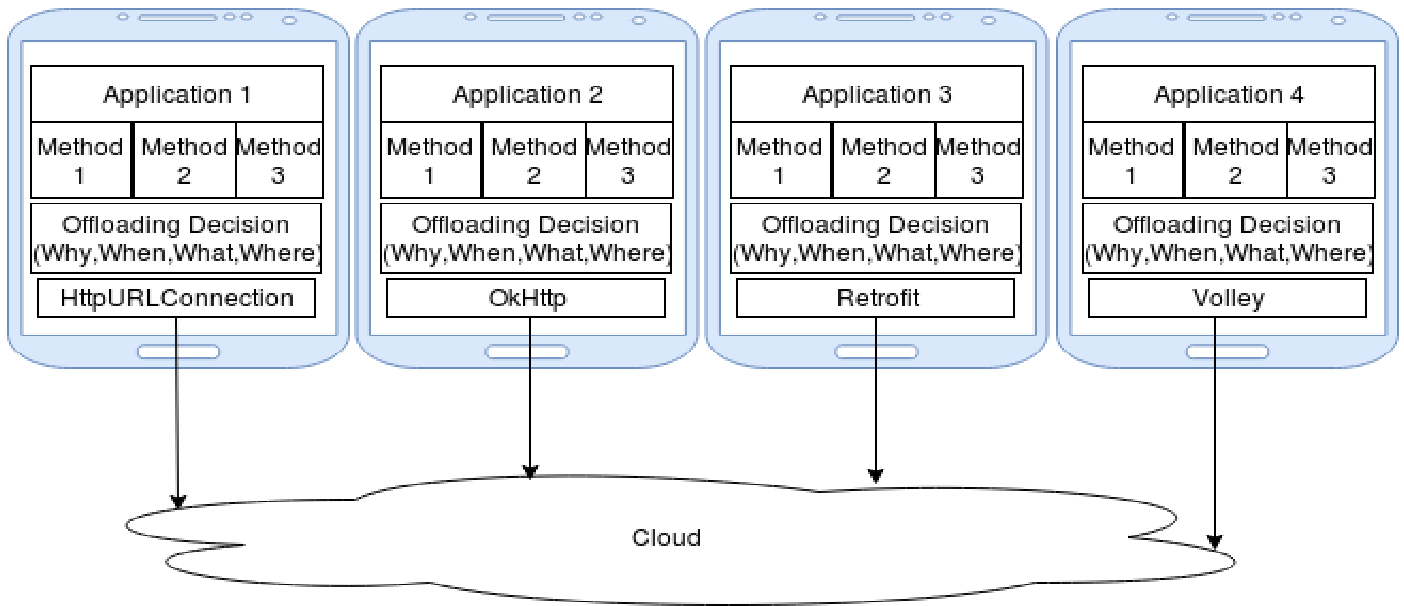


Fig. 2. Current Offloading Trend : Offloaded task or data is send to the cloud using any predefined network library.

offloading libraries that are currently supported by the Android OS and understand their behavior for the above factors.

As discussed before, researchers have rigorously studied the code profilers [17], network profilers [18], hardware and software profilers [19] to propose models and frameworks for addressing the *why, what, when*, and *where* challenges of offloading. However, in this paper, we are trying to explore the effect of the network libraries on offloading by analyzing the existing technologies using real implementation. The idea is to evaluate the effect of the network libraries on offloading. We believe that *how-to-offload* is an important factor in offloading that requires a more thorough investigation and careful evaluation. Without loss of generality, we abstract the problem of understanding the behavior of the libraries independent of any application in terms of the following factors:

- **Synchronous or Asynchronous Offloading:** An offloading operation can either be executed synchronously or asynchronously dependent upon the requirements of an application. Synchronous execution means the execution in a series. For example, in a chess game, a player makes the next move when the opponent turn is over. Asynchronous execution means to split the problem into multiple tasks and process them independently. For example, *discussion forums* where every user can post their views independent of any other user.
- **Data Size:** Library performance also depends on the amount of data an application needs to offload. For some application, we need to transfer only a small amount of data such as program files for execution. On the other hand, some application may require transferring large files such as video or images for analysis. Hence, there is a need to decide which of the above libraries is best suited for transferring a particular data size.
- **Network Medium:** The performance of the library also depends upon the communication media used for transferring the data, such as Wi-Fi or 4G.
- **Hardware/Software:** The effect of hardware configuration and operating system of the mobile devices on these libraries.

To analyze the performance of the networking libraries, we develop android applications for implementing these libraries. We consider a scenario, where a network library needs to offload data to the cloud, either in synchronous or asynchronous mode, via its underlay communication media. Evaluation is done on a test-bed, involving different file sizes, code execution, network medium and mobile devices of various configuration. For this evaluation, we spawn a virtual machine in Amazon Web Services (AWS) cloud to study and analyze the behaviors of these libraries in a real environment.

The rest of the paper is structured as follows. Section II provides the background of offloading frameworks and techniques. The terminologies used in our paper are described in Section III. In Section IV, we describe the overview of different networking libraries of Android OS and implementation detail of analyzing *how* to offload aspect. Section V presents the experimental setup used in the performance analysis of the various networking libraries. The results obtained from the performance evaluation are given in Section VI. In Section VII, we discuss the results obtained from the performance evaluation of offloading libraries in detail. Finally, we conclude our paper and discuss future works in Section VIII.

II. Related Work

In this section, we present a review of the proposed offloading frameworks and techniques. In [9], authors present a *MAUI* framework using a strategy based on code annotations to determine the computation intensive methods that can be offloaded. The main aim of this framework is to save the

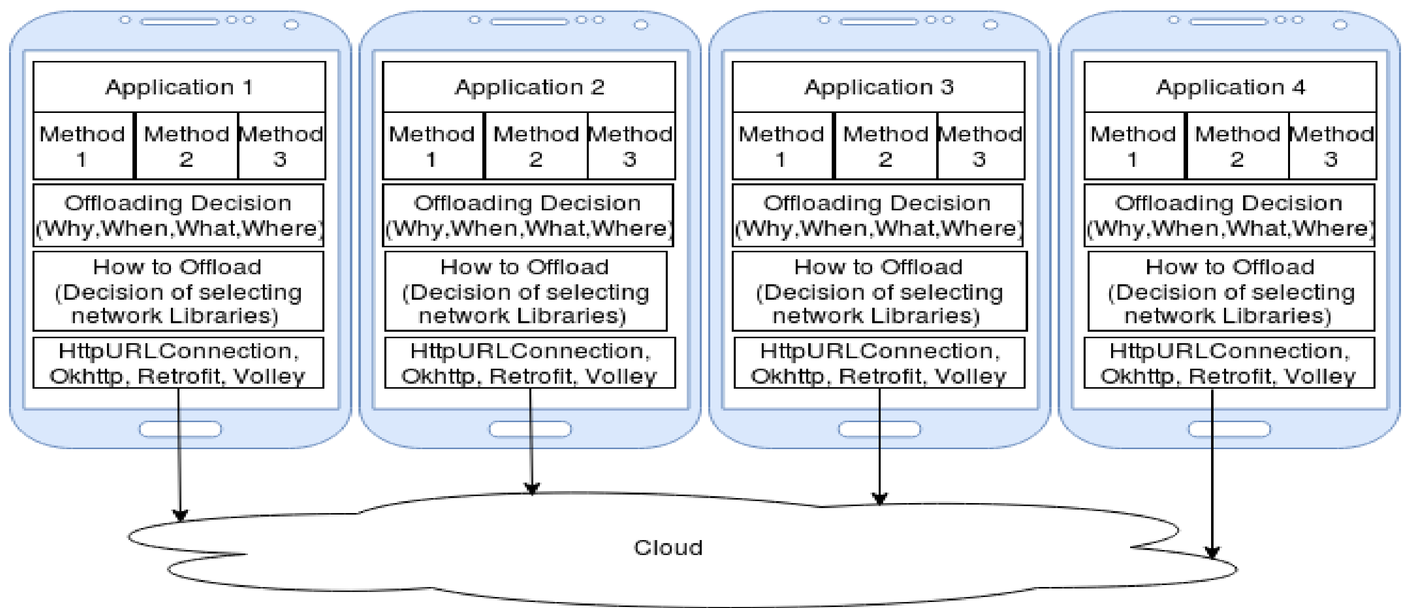


Fig. 3. How to Offload: Adaptive network library selection depending upon the parameters like task execution, data size, network medium and H/W & S/W of smartphones.

energy of mobile devices by analyzing the computation intensive code using MAUI profiler which minimizes the burden of program partitioning on the programmer. Authors evaluate the MAUI's energy consumption and performance benefits for different applications.

Clone Cloud [12] is another prominent framework for code offloading. In this model, a clone of the smartphone is maintained in the cloud that is synchronized with the user's smartphone before offloading. The framework partitions the application utilizing static and dynamic profiling to optimize execution time and energy. In this article, authors have tested their model for the different applications and shown a relative improvement in execution speed and energy consumption of the mobile devices.

In [7], authors present *ThinkAir* framework to perform on-demand resource allocation. It exploits scalable resources of cloud by dynamically creating, resuming, and destroying virtual machines (VMs) whenever required for parallel execution of offloaded code to reduce execution time. In this paper, the authors analyze the execution time and battery consumption of applications over different networks for evaluating the framework.

In [20], authors propose *ENDA*, which is a three-tier offloading architecture involving smartphones, cloudlets, and cloud interacting among themselves to consider user mobility, network performance, and server load to make efficient offloading decisions. Authors design a greedy search algorithm to predict the user movement and select the energy efficient Wi-Fi access point for offloading. The main focus of ENDA is to generate optimal offloading decision by considering the user mobility and unstable network quality.

[21] discuss the *COSMOS* framework to provide offloading as-a-service to smart mobile devices. COSMOS acts as an intermediary between cloud and smartphone for cost-effective scheduling and allocation of the cloud resources, after receiving the offloading request from the mobile devices. The framework enhances the speed of mobile computation while at the same time reduce the cost of leasing cloud resources.

Authors in [22], puts forward a context-sensitive offloading decision algorithm to decide the network medium and cloud resource, including the resources of local mobile device cloud, cloudlet, public cloud for offloading at runtime based on the device context to improve the performance. In [23], an offloading strategy *Cuckoo* is introduced by authors. The main task of Cuckoo is to save battery life and minimize cost using static and dynamic profiling. In this paper, authors propose a skyline-based online resource scheduling to satisfy the offloading demands.

In [18], authors present *SIMDOM* offloading framework which translates the computation and resource intensive Single Instruction, Multiple Data (SIMD) instructions in a cloud or edge environments. The framework performs vector-to-vector instruction mappings to translate the ARM SIMD intrinsic instructions to x86 SIMD intrinsic instructions, so that mobile platform application can easily be executed on heterogeneous machines in a cloud or edge server without any modification. Offloading decision is taken by an offload manager using the values received from the application, energy, and network profilers.

In [11], authors propose *EMCO*, which uses crowdsensing to improve offloading decisions rather than profiling different parameters of individual devices. EMCO utilizes crowd sensed evidence traces as a novel mechanism for improving the performance of offloading systems. In [17], authors present MobiCOP, an offloading platform solution which is fully self-contained in a library format. Any Android software can integrate with MobiCOP without requiring extraneous third-party tools. The authors focus on the real-life implementation of the offloading solution irrespective of any customized OS versions.

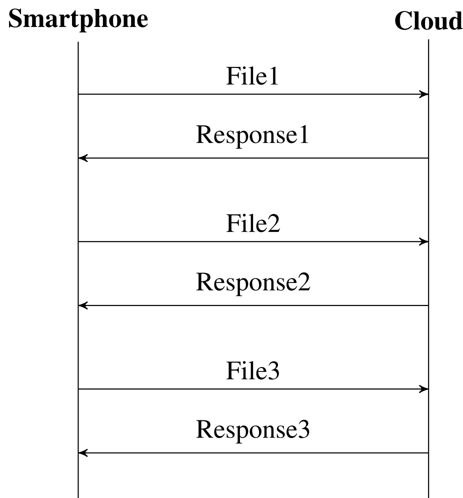


Fig. 4. Synchronous Computation/Data Transfer

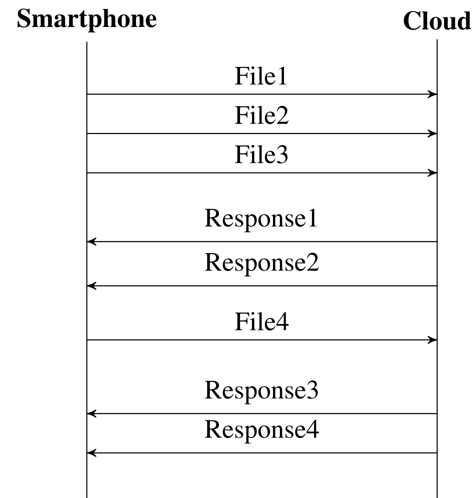


Fig. 5. Asynchronous Computation/Data Transfer

Most papers discussed above evaluate the performance of a framework by analyzing the total time required for executing a task and battery consumption when offloading over the different network medium. However, in this paper, we focus on the network libraries that are often used for offloading the computation tasks to the cloud. We intend to investigate the need for adaptive network library selection procedure, depending upon dynamic parameters like data size, execution mode, communication media, hardware and software of the mobiles devices.

III. Terminologies

1) Data vs Code Offloading: The smartphones can either offload code or data to the cloud. Sensors are the most prominent source of data generation. The data generation rate of the sensors can vary depending upon the requirements of the application. However, most of the big data analytics and machine learning programs are executed in the cloud, which requires the user to transfer a large amount of data from its smart devices to the cloud, also known as *data offloading*.

The data-intensive applications and sensors often store huge amount of data in the cloud and downloading the data to a local server for processing may not be efficient. A user needs to send the program file or code from smart device to cloud for performing the particular computation. After processing data, the result is revert to the user. This strategy requires only transferring a small amount of data, i.e., program code and result from the server, also known as *code offloading*. The problem of an efficient code offloading can be complex, particularly if it involves multiple servers [24] or applications.

2) Synchronous vs Asynchronous Offloading: There are two ways in which code/data can be offloaded for execution on the cloud *synchronously* or *asynchronously*. The synchronous transfer used in the applications that require serial execution of a particular task. In other words, the tasks are executed one after the other. In synchronous execution, the output received from the executed code can is used as an input for another program, i.e., the tasks can be dependent on each other. Synchronous execution is also utilized in situations where the tasks are frequently accessing a shared memory location [25]. A real-life example of synchronous execution is communication over walkie-talkie where a person responds when the other person's message is finished. Another example, a chess game, a player will make the next move when the opponent turn is over. Fig. 4 shows the sequence diagram of the synchronous transfer of files from a mobile device to the cloud. Next file is started to upload when the successful acknowledgment or result of the previous file is received from the cloud.

Similarly, asynchronous offloading used in situations where the task is independent of each other and don't access shared memory. In other words, the result of the offloaded task is not required by other tasks for their execution. Asynchronous execution split up the problem into multiple tasks and process them independently [26], [27]. A real-life example of asynchronous execution is *discussion forums* where every user can post their views independent of any other user. Another example is communication over mobile networks where persons listen and respond when talking to the other person simultaneously. Fig. 5 shows the sequence diagram of the asynchronous transfer of files from a mobile device to the cloud. Files are uploaded one after another consecutively before receiving the response from the cloud server.

IV. Offloading Libraries

In this section, we provide a detailed discussion of four different network libraries that are commonly used for exchanging data from the cloud.

A. HttpURLConnection

HttpURLConnection is an abstract class of JAVA extended from the URLConnection class. Developers popularly use it for exchanging data from the web servers. As the name suggests, it works on HTTP protocol and contains additional HTTP specific features. A single instance of HttpURLConnection is used to make a single request from the HTTP server. HttpURLConnection can be used only for the synchronous networking calls; it does not support asynchronous calls.

Before the introduction of other networking libraries, it was officially suggested by the Android developing team to use HttpURLConnection [28] for the networking purposes.

To use the HttpURLConnection class for uploading data from a client or smart-device to the server is started by obtaining a new HttpURLConnection by calling *URL.openConnection()* and casting the result to *HttpURLConnection* and configure the connection for output using *setDoOutput(true)*. If the size of the file or data is known in advance we can call *setFixedLengthStreamingMode(int)* or *setChunkedStreamingMode(int)* when it is not. Below we give the abstract code to perform an upload using HttpURLConnection class:

```
URL url = new URL(uploadServerUrl);
HttpURLConnection urlConnection =
    (HttpURLConnection) url.openConnection();
  try {
    urlConnection.setDoOutput(true);
    urlConnection.setChunkedStreamingMode(0);

    OutputStream out = new
        BufferedOutputStream
        (urlConnection.getOutputStream());
    writeStream(out);

    InputStream in = new BufferedInputStream
        (urlConnection.getInputStream());
    readStream(in);
  } finally {
    urlConnection.disconnect();
  }
```

Below is the abstract code for retrieving the result or file from the server using HttpURLConnection class:

```
URL url = new URL(sourceFileUrl);
  HttpURLConnection urlConnection =
      (HttpURLConnection)
      url.openConnection();
  try {
    InputStream in = new BufferedInputStream
        (urlConnection.getInputStream());
    readStream(in);
  } finally {
    urlConnection.disconnect();
  }
```

B. OkHttp

OkHttp networking library is an open source project which is introduced by *Square* [29]. OkHttp is an efficient HTTP client which supports HTTP, HTTP 2.0 and SPDY protocols. OkHttp multiplex several HTTP requests over one socket connection. OkHttp is a powerful networking tool which does not require any REST library and its also support both synchronous and asynchronous networking calls. It also provides the caching mechanism to cache the response from the server to avoid repeated network request.

Below we give an abstract code to perform a file upload synchronously and asynchronously. For synchronous network call, create a *call* object using *client* and use the *execute* method. The synchronous request should be executed on a background thread; otherwise, it gives network error. For asynchronous network call, execute method is replaced with the *enqueue* method. There is no need for background thread for making asynchronous network calls.

```
OkHttpClient client = new OkHttpClient();
RequestBody file_body = RequestBody.create
    (MediaType.parse(content_type), file);
RequestBody request_body = new
    MultipartBody.Builder()
                .setType(MultipartBody.FORM)
                .addFormDataPart ("name",
                    file_name, file_body)
                .build();
Request request = new Request.Builder()
                .url(ServerUrl)
                .post(request_body)
                .build();
// For Synchronous Calls
Response response =
    client.newCall(request).execute();
// For Asynchronous Calls
client.newCall(request).enqueue(new
    Callback() {
   public void onFailure(Call
        call,IOException e){
   }
   public void onResponse(Call call, final
        Response response) throws IOException {
       // do something wih the result
   }
}
```

Similarly, files can be downloaded from the cloud in synchronous and asynchronous manner using.

```
OkHttpClient client = new OkHttpClient();
Request request = new
    Request.Builder().url(file_url).build();
// For Synchronous Calls
try {
  Response response =
      client.newCall(request).execute();
  write(fileTodisk);
}catch (Exception e) {
  e.printStackTrace();
}
// For Asynchronous Calls
try {
  Response response =
      client.newCall(request).enqueue(new
      Callback() {
   public void onFailure(Call
        call,IOException e){
   }
   public void onResponse(Call call, final
        Response response) throws IOException {
       // do something wih the result
   }
```

```
}
```

C. Volley

Volley is a HTTP networking library introduced by *Google* in *Google I/O 2013*. Volley provides many powerful networking tools out of the box for the users. Some of the prominent features are multiple concurrent network request, automatic scheduling, and prioritization of the network request, cancellation of single or blocks of requests and provide effective memory response caching. Volley is easy to code, and it fetched data asynchronously from the network [30]. Here, we give an abstract code of sending a file to the server using Volley.

```
// Instantiate the RequestQueue
RequestQueue queue =
    Volley.newRequestQueue(this);
String url ="http://www.serverip.com";

// Request a string response from the
    provided URL.
MultiPartRequest request = new
    SimpleMultiPartRequest(Request.Method.GET,
    serverUrl,
          new Response.Listener<String>() {
   @Override
   public void onResponse(String response) {
   }
}, new Response.ErrorListener() {
   @Override
   public void onErrorResponse(VolleyError
       error) {
   }
});
//Add the request to the RequestQueue
request.addFile("name", file);
queue.add(request);
```

Below is the code for downloading a file from the server using Volley:

```
RequestQueue queue = Volley.newRequestQueue();
InputStreamVolleyRequest request = new
    InputStreamVolleyRequest(Request.Method.GET,
    fileUrl, this,this, null);
queue.add(request);
@Override
public void onErrorResponse(VolleyError
    error) {
}
@Override
public void onResponse(byte[] response) {
  writeFileToDisk();
}
```

D. Retrofit

Retrofit is type-safe and one of the most popular HTTP client for Android by *Square*. It is very easy to use and convert the HTTP API into Java interface. It performs network function using REST based web services. Retrofit support both synchronous and asynchronous network request to the remote web server. It also provides a caching mechanism for repeated network request [31]. Retrofit converts HTTP API into Java interface which helps to treat your network calls as simple Java method calls. Below is the abstract code for uploading files to the server in both synchronous and asynchronous manner:

```
MultipartBody.Part filePart =
    MultipartBody.Part .createFormData("name",
    file_name, RequestBody
    .create(MediaType.parse("*/*"), file));
Retrofit.Builder builder = new
    Retrofit.Builder()
                 .baseUrl(serverUrl);
Retrofit retrofit = builder.build();
Upload api = retrofit.create(Upload.class);
Call<ResponseBody> call=
    api.uploadAttachment(filePart);
//For Synchronous Calls
call.execute();
//For Asynchronous Calls
call.enqueue(new Callback<ResponseBody>() {
@Override
public void onResponse(Call<ResponseBody>
    call, Response<ResponseBody> response) {}
@Override
public void onFailure(Call<ResponseBody>
    call, Throwable t) {}
}
//Interface for File Upload
interface Upload {
   @Multipart
   @POST("upload.php")
   Call<ResponseBody> uploadAttachment(
         @Part MultipartBody.Part filePart);
}
```

Below is the abstract code for downloading files from the server in both synchronous and asynchronous manner:

```
Retrofit.Builder builder = new
    Retrofit.Builder()
                 .baseUrl(serverUrl);
Retrofit retrofit = builder.build();
Download api =
    retrofit.create(Download.class);
Call<ResponseBody> call =
    api.downloadFile(fileURL);
//For Synchronous Calls
Response<ResponseBody> response =
    call.execute();
writeFileToDisk();
//For Asynchronous Calls
call.enqueue(new Callback<ResponseBody>() {
@Override
public void onResponse(Call<ResponseBody>
    call, Response<ResponseBody> response) {
  writeFileToDisk(); }
@Override
public void onResponse(Call<ResponseBody>
    call, Response<ResponseBody> response) {
```

TABLE I. AWS EC2 INSTANCE CONFIGURATION

EC2 Instances	CPU	Memory	Memory
t2.micro	1 GHz	1 GB	8 GB

TABLE II. OFFLOADING LIBRARIES

S.No	Synchronous Transfer	Asynchronous Transfer
1	HttpUrlConnection	Volley
2	OkHttp Synchronous	OkHttp Asynchronous
3	Retrofit Synchronous	Retrofit Asynchronous

```
  writeFileToDisk(); }
//Interface for File Upload
public interface Download {
   @GET
   Call<ResponseBody> downloadFile(@Url
       String url);
}
```

V. EXPERIMENTAL SETUP

To understand the performance, we develop an Android application to implement different network libraries. We study the behavior of the libraries under different parameters like code execution, data size, wireless media, and mobile devices. The evaluation is conducted on the WiFi and 4G networks, using two smartphones of different hardware and software configurations. The experiments evaluate the performance of both synchronous and asynchronous libraries by sending the files of different sizes to the cloud through WiFi and 4G networks. For real evaluation, we spawn a virtual machine or a *Elastic Compute Cloud (EC2)* instance in AWS cloud, so that the behavior of these libraries is studied and analyzed in a real environment. The configuration details of an (EC2) instance is given in Table I. We place our virtual machine in the US-West (Oregon) data center region. The virtual machine is intentionally placed very far, to consider the worst case scenario and evaluate the libraries under varied network traffic conditions. We conduct the experiments for a week, at different times during day and night. The performance of libraries is analyzed regarding battery consumption and network delay incurred due to the effect of various factors such as file size, network media, hardware and software of the smartphones. The libraries used for synchronous or asynchronous data transfer from mobile device to the cloud are mentioned in Table II.

Depending upon the nature of the applications, we may need to upload data of different size ranging from few bytes to MB's, in our paper we consider the files size ranging from 200 bytes - 8 MB. In the following discussion, the term "data" or "file" is used to represent both code and data offloading, as discussed previously in Section III. Code offloading requires sending files of small size, whereas data offloading transfers a large amount of data for storage in the cloud. While testing the libraries for particular file size, we send multiple copies of the same file to the cloud to negate the effect of the network on the performance of a library at a particular instant. For synchronous transmission, we send the files one-by-one after receiving the response from the server; whereas for asynchronous transmission, we send the files in parallel without waiting for the response from the cloud.

TABLE III. DEVICE CONFIGURATION

Name	OS	CPU	RAM
Smartphone-1	Android v6.0.1	Quad-core 2.5 GHz Krait 400	3GB
Smartphone-2	Android v5.1	1.0 GHz quad core MediaTek	1GB

We perform the same experiments in different wireless networks, i.e., WiFi and 4G. Moreover, we analyze the effect of hardware and software of the smart devices on the performance of libraries as there is a huge difference in the hardware capabilities of different smart devices. The configurations of the smartphones used in our evaluation are in Table III. We evaluate all libraries on both mobiles at the same time by executing the process of uploading and downloading on different threads simultaneously. The overall result is averaged.

VI. PERFORMANCE EVALUATION

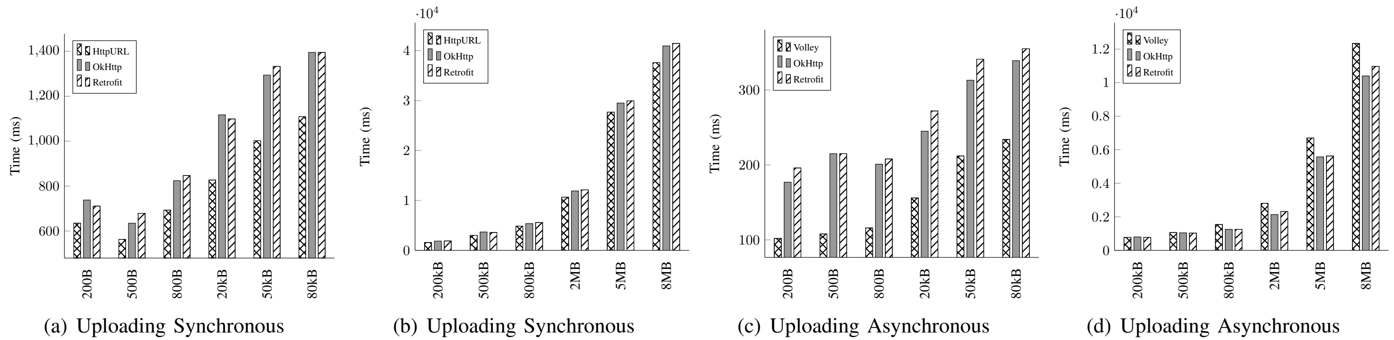
In the following section, we first introduce the parameters used for the performance evaluation and later, we explain the experimental results that are received after the analysis of the network libraries.

- **Total Delay:** Total delay is the time required for uploading/downloading files to and from a mobile device to the AWS cloud. This includes the time required for reading the file from secondary storage to the internal buffer, transmission time and ACK time from the cloud. This parameter is very crucial for time-sensitive applications.
- **Success rate:** The success rate is calculated as a ratio of the total successful acknowledgments received by the total number of files sent. This parameter shows the reliability of a library for an offloading application.
- **Battery utilization:** One of the major goals of offloading is to save the energy of mobile devices by migrating heavy computation to the cloud. This parameter evaluates the battery consumption of networking libraries in different circumstances.

A. Analyzing Total Delay

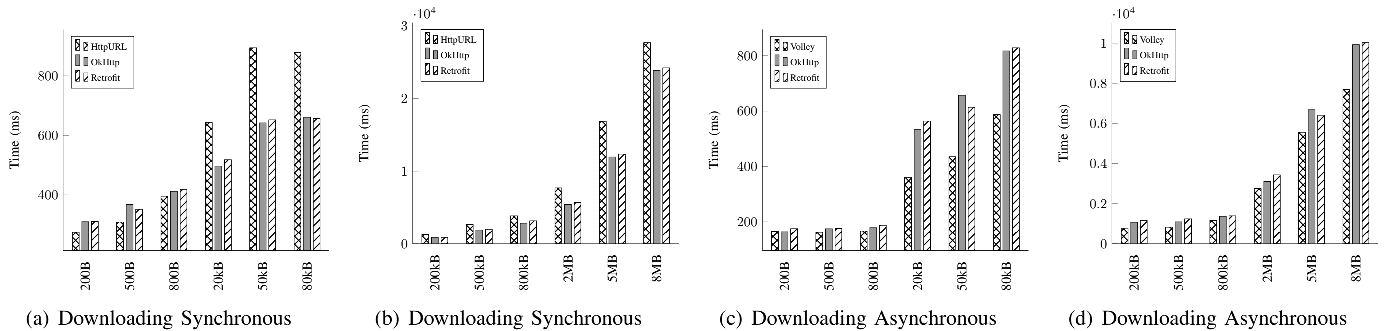
1) Upload Performance: Fig. 6 shows the upload timing for different file sizes on WiFi network for both synchronous and asynchronous transmissions. Among synchronous libraries, the HttpURL performs better than OkHttp and Retrofit for small file sizes ($<$ 80 kB). As shown in Fig. 6(a), the difference in network delay between HttpURL and OkHttp/ Retrofit is around $100 - 300$ ms. However, as the file size increases (between 200 kB - 8 MB), the difference becomes more prominent and reaches $3000 - 4000$ ms, Fig. 6(b). Although the time difference for small file size is not much, if we have a large number of small files to offload or working on time-sensitive applications, the overall difference is quite significant.

For asynchronous transmission, the performance of Volley is better for small file size in comparison to OkHttp and Retrofit. The difference in the network delay between Volley and the other two libraries is around 100 ms, Fig. 6(c). However, it is not true in case of large files, the performance of OkHttp and Retrofit improves as the file size increases ($>$ 200



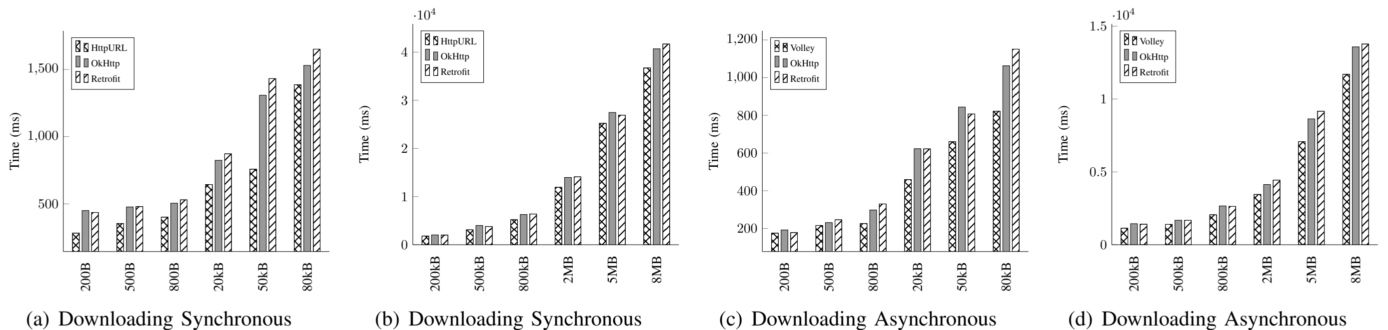
(a) Uploading Synchronous (b) Uploading Synchronous (c) Uploading Asynchronous (d) Uploading Asynchronous

Fig. 6. Time delay comparison of networking libraries when uploading from smartphone-1 using WiFi.



(a) Downloading Synchronous (b) Downloading Synchronous (c) Downloading Asynchronous (d) Downloading Asynchronous

Fig. 7. Time delay comparison of networking libraries when downloading from smartphone-1 using WiFi.



(a) Downloading Synchronous (b) Downloading Synchronous (c) Downloading Asynchronous (d) Downloading Asynchronous

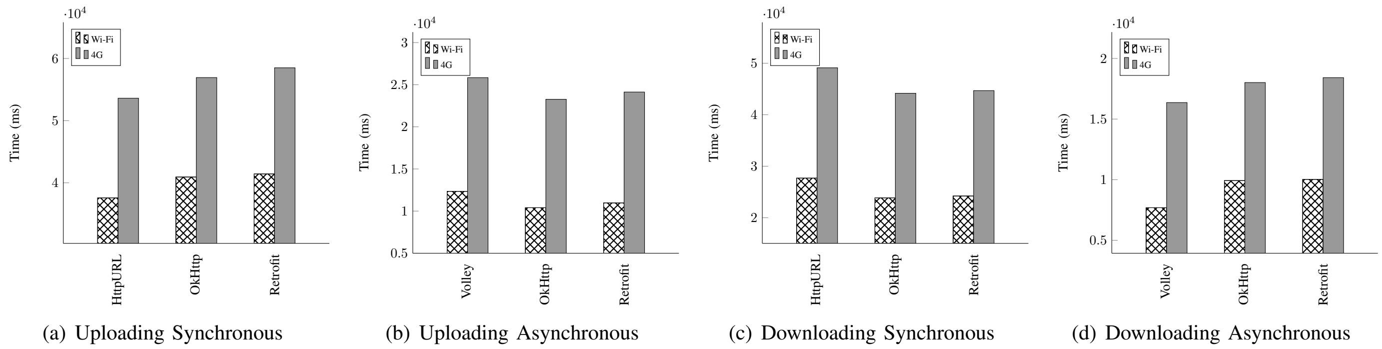
Fig. 8. Time delay comparison of networking libraries when downloading from smartphone-2 using WiFi.

kB). The difference in network delay between OkHttp and Volley for large files size (8 MB) rise to 2000 ms, Fig. 6(d).

Our experiment shows a similar pattern for both synchronous and asynchronous transmission in the 4G network for smartphone-1. For smartphone-2, the pattern is similar to that of smartphone-1 for both WiFi and 4G network medium. In the case of smartphone-2, the network delay for synchronous and asynchronous transmission rise by $600-700$ and $300-350$ ms respectively in a WiFi network. Similarly, the network delay increases by $1000-1200$ & $600-700$ ms for both synchronous and asynchronous transmission in the 4G network. This increment in network delay is due to the difference in hardware and software configuration of both the smartphones, discussed later in Section VI-A4.

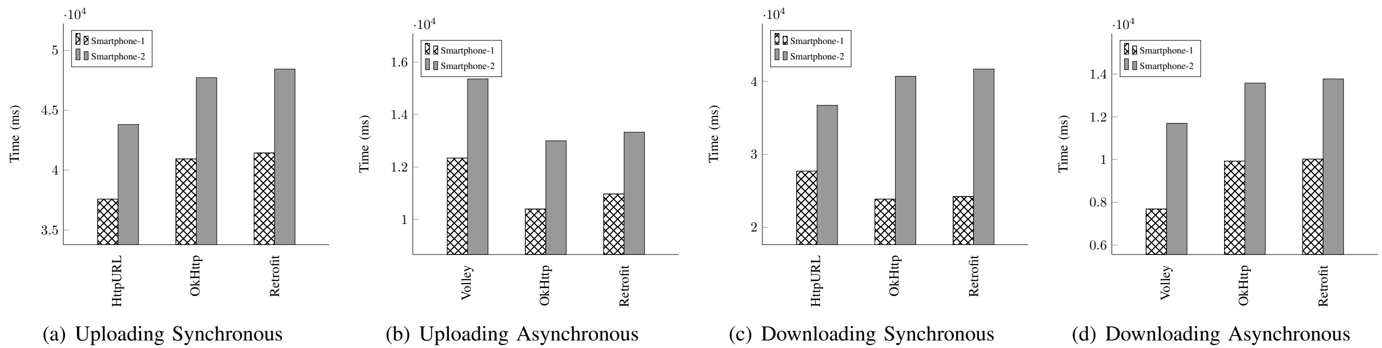
2) Download Performance: Fig. 7 shows the downloading performance of both synchronous and asynchronous offloading libraries on smartphone-1 using WiFi network. Our experiment shows a similar pattern for the 4G network. Fig. 7(a) and 7(b) shows that OkHttp and Retrofit performs better than HttpURL. For small file sizes, the network delay difference between OkHttp/Retrofit and HttpURL is around $100 - 200$ ms. This difference can prove to be quite significant for the applications downloading a large number of files or time-sensitive result from the cloud. However, as the file size increases, the difference gets more noticeable and reaches to $3000 - 4000$ ms for large file size (8 MB). Similarly, for asynchronous transmissions, as shown in Fig. 7(c) and 7(d), the performance of Volley is better than OkHttp and Retrofit. The network delay difference of Volley and OkHttp for small files is around 100 ms, and for large file size, the difference is around 2000 ms.

For smartphone-2, the network delay pattern in WiFi and 4G is similar to the smartphone-1, but in case of synchronous transmission in WiFi network, HttpURL performs better than OkHttp and Retrofit. The contrast in the performance of the libraries can be seen in Fig. 7(a), 8(a) & 7(b), 8(b). The results suggest that for downloading files/output synchronously from



(a) Uploading Synchronous (b) Uploading Asynchronous (c) Downloading Synchronous (d) Downloading Asynchronous

Fig. 9. Time delay comparison of WiFi & 4G for different libraries using smartphone-1



(a) Uploading Synchronous (b) Uploading Asynchronous (c) Downloading Synchronous (d) Downloading Asynchronous

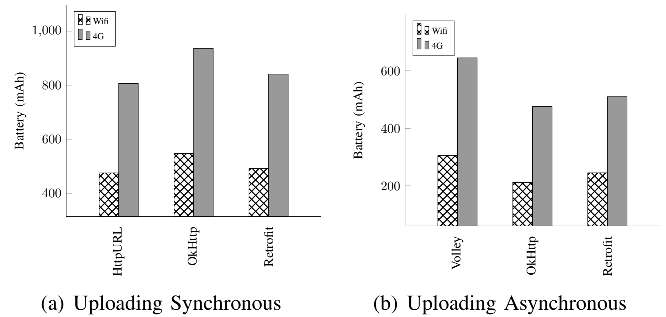
Fig. 10. Time delay comparison of smartphone-1 and smartphone-2 on WiFi network

cloud to a high-end smartphone, OkHttp and Retrofit better than HttpURL. But, for low-end smartphones, the HttpURL performance is better than OkHttp and Retrofit.

3) Network comparison: In this section, we compare the network delay of different network medium, i.e., WiFi and 4G for both uploading and downloading. We compare the network delay of both the network media by sending the same file (8 MB) using smartphone-1. The result is shown in Fig. 9 and similar trend follows for smartphone-2. From the results, we find that the performance of WiFi is better than the 4G network. The delay difference between both the network for uploading 8 MB file in synchronous mode is $1300 - 1700$ ms, and for asynchronous mode, the delay difference is $1100 - 1300$. Similarly, the network delay difference in WiFi and 4G network for downloading large data (8 MB) from the cloud in synchronous mode is $2000 - 2300$ ms and for asynchronous mode is $800 - 1000$ ms.

The primary advantage of 4G is near-ubiquitous coverage over WiFi. The recent studies have shown that the WiFi offers higher and consistent throughput whereas round-trip times of 4G are lengthy and bandwidth is limited [32]. In our experiment, we found that the round-trip time of WiFi and 4G network to the virtual machine setup in the AWS cloud lies in the range 300 - 350 ms and 480 - 550 ms respectively.

4) Hardware and Software comparison: Finally, we study the effect of hardware and software on the performance of the offloading libraries. We compare the uploading and downloading network delay analysis of both the smartphones using 8 MB file on the WiFi network. The result for the WiFi is shown in Fig. 10, and a similar trend follows for the 4G network. In the case of offloading, the time delay for synchronous libraries is between $600 - 700$ ms, and for synchronous libraries, it is around $300 - 350$ ms. For downloading the result or data in synchronously, the delay is around $900 - 1600$ ms, and for asynchronous transmission, it is around $800 - 1000$ ms.



(a) Uploading Synchronous (b) Uploading Asynchronous

Fig. 11. Battery Consumption Analysis for smartphone-1 on WiFi

Our analysis shows that the performance of the networking libraries for a high-end smartphone (Smartphone-1) is better compared to the low-end smartphone (Smartphone-2). This difference in the performance of libraries is due to the difference in the hardware and software of the smartphones. Hardware differences in smartphones like WiFi antenna, memory (required for buffering) and processing power plays an important role in the performance of the libraries. Besides, smartphones with latest OS can manage the resources more efficiently, in comparison to an older version of the OS.

B. Success Rate Analysis

Next, we try to evaluate the success rate of the libraries, i.e., the number of successful transmissions by sending 500 files (i.e., 1 MB $\times$ 500 = 500 MB) to the cloud in both synchronous and asynchronous mode. The success rate is very an important parameter for understanding the reliability of a library while transferring sensitive data to the cloud. We find that the success rate of synchronous libraries Table IV is better than asynchronous libraries Table V. The success rate of the 4G network is less than the WiFi network, which can be due to the large number of users and variations in the network traffic in the 4G network. Overall the reliability of Retrofit is better than the other libraries for both synchronous and asynchronous scenarios on both networks. The success rate of Retrofit is almost 100% in all the scenarios.

C. Battery Consumption Analysis

We also study the energy consumption of the offloading libraries on WiFi and 4G networks, which is an important concern for the resource-constrained mobile devices. In this, we used the power-save mode (PSM) for the results presented in Fig. 11. In power save mode, the smartphone's WiFi radio wakes up only when it has to transmit data and once every 100 ms when it checks whether there is any incoming data from the access point. Fig. 11, also shows that the battery consumption by the asynchronous libraries is almost half compared to synchronous libraries. Also, the battery consumption in 4G is almost twice as compared to the WiFi network, which is due to the lengthy RTT and limited bandwidth of the 4G network as compared to WiFi. Similar, results are also found for the smartphone-2.

TABLE IV. SUCCESS RATE OF SYNCHRONOUS LIBRARIES

S.No	Name	Success Rate-WiFi	Success Rate-4G
1	HttpURL	100%	99%
2	OkHttp	100%	99%
3	Retrofit	100%	100%

TABLE V. SUCCESS RATE OF ASYNCHRONOUS LIBRARIES

S.No	Name	Success Rate-WiFi	Success Rate-4G
1	Volley	99%	97%
2	OkHttp	100%	99%
3	Retrofit	100%	100%

VII. DISCUSSION

In this section, we briefly discuss the role of networking libraries in offloading computation intensive tasks form the smart devices and the highlight the requirement of a framework for adaptive library selection based on the local dynamics of the smartphones.

- From experiments, we find that in the case of synchronous offloading, HttpUrl performs better than Retrofit and OkHttp for small file sizes. However, as the file size increases, OkHttp performs better than the two. This is true for both Wifi and 4G networks. We notice an exception for the low-end smartphones operating on WiFi networks, where HttpUrl shows a better performance for large file sizes as well.
- Offloading in asynchronous mode is also library dependent. In the case of asynchronous uploading, Volley and OkHttp perform better for small and large files, respectively. However, for asynchronous download from the cloud, Volley performs better than OkHttp and Retrofit for both small and large files.
- One of the main objectives of offloading is to save battery in smartphones. In this context, HttpUrl shows better performance for synchronous transmission, whereas OkHttp is more energy efficient for asynchronous transmissions.
- Although the offloading delay for synchronous transmissions is greater than that of asynchronous transmissions, the opposite is true for offloading reliability. Retrofit shows better reliability for both synchronous and asynchronous transmissions in WiFi and 4G networks.

Our above discussion shows that the performance of the networking libraries depends upon the parameters like data size, network medium, hardware and software of mobile devices. An application with a predefined network library may not achieve the desired offloading performance for all mobile devices. So, an adaptive offloading framework is required for providing better QoS to the user.

VIII. CONCLUSION

In this paper, we present a comprehensive analysis of HttpUrl, OkHttp, Retrofit and Volley networking libraries that are commonly used for offloading data from the mobile devices. The main objective of this work is to understand "how to offload" data in a real environment, to optimize the performance of an offloading model. In this paper, we investigate the performance of the libraries for parameters like code execution, data size, network medium, hardware and software of mobile devices. Our evaluation shows that, depending upon the nature of the application, available resources and offloading goals like execution speed, reducing energy consumption, reliability of data transmission, an adaptive network library selection framework can be developed for offloading computational tasks to the cloud. In the future, we intend to propose an offloading framework for adaptive network library selection depending upon the above criteria.

REFERENCES

[1] B. G. Rodriguez-Santana, A. M. Viveros, B. E. Carvajal-Gamez, and D. C. Trejo-Osorio, "Mobile computation offloading architecture for mobile augmented reality, case study: Visualization of cetacean skeleton," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 1, 2016. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2016.070190

[2] A. Banerjee, F. Sufyanf, M. S. Nayel, and S. Sagar, "Centralized framework for controlling heterogeneous appliances in a smart home environment," in *2018 International Conference on Information and Computer Technologies (ICICT)*, March 2018, pp. 78–82.

[3] R. K. Balan and J. Flinn, "Cyber foraging: Fifteen years later," *IEEE Pervasive Computing*, vol. 16, no. 3, pp. 24–30, 2017.

[4] E. Meskar, T. D. Todd, D. Zhao, and G. Karakostas, "Energy aware offloading for competing users on a shared communication channel," *IEEE Transactions on Mobile Computing*, vol. 16, no. 1, pp. 87–96, Jan 2017.

[5] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya, "Mobile code offloading: from concept to practice and beyond," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 80–88, March 2015.

[6] C. You, K. Huang, H. Chae, and B. H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp. 1397–1411, March 2017.

[7] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 Proceedings IEEE INFOCOM*, March 2012, pp. 945–953.

[8] H. Elazhary, S. Aloraini, and R. Aljuraid, "Context-aware mobile application task offloading to the cloud," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 5, 2017. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2017.080547

[9] 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*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 49–62. [Online]. Available: http://doi.acm.org/10.1145/1814433.1814441

[10] J. Wang, J. Peng, Y. Wei, D. Liu, and J. Fu, "Adaptive application offloading decision and transmission scheduling for mobile cloud computing," *China Communications*, vol. 14, no. 3, pp. 169–181, March 2017.

[11] H. Flores, P. Hui, P. Nurmi, E. Lagerspetz, S. Tarkoma, J. Manner, V. Kostakos, Y. Li, and X. Su, "Evidence-aware mobile computational offloading," *IEEE Transactions on Mobile Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[12] 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*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 301–314. [Online]. Available: http://doi.acm.org/10.1145/1966445.1966473

[13] A. Saarinen, M. Siekkinen, Y. Xiao, J. K. Nurminen, M. Kemppainen, and P. Hui, "Can offloading save energy for popular apps?" in *Proceedings of the Seventh ACM International Workshop on Mobility in the Evolving Internet Architecture*, ser. MobiArch '12. New York, NY, USA: ACM, 2012, pp. 3–10. [Online]. Available: http://doi.acm.org/10.1145/2348676.2348680

[14] B. S. Rawal, "Proxy re-encryption architect for storing and sharing of cloud contents," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 0, no. 0, pp. 1–17, 2018. [Online]. Available: https://doi.org/10.1080/17445760.2018.1439491

[15] G. Andriani, E. Godoy, G. Koslovski, R. Obelheiro, and M. Pillon, "An architecture for synchronising cloud file storage and organisation repositories," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 0, no. 0, pp. 1–17, 2018. [Online]. Available: https://doi.org/10.1080/17445760.2017.1422500

[16] H. Wu, "Multi-objective decision-making for mobile cloud offloading: A survey," *IEEE Access*, vol. 6, pp. 3962–3976, 2018.

[17] J. I. Benedetto, A. Neyem, J. Navon, and G. Valenzuela, "Rethinking the mobile code offloading paradigm: From concept to practice," in *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*, ser. MOBILESoft '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 63–67. [Online]. Available: https://doi.org/10.1109/MOBILESoft.2017.20

[18] J. Shuja, A. Gani, K. Ko, K. So, S. Mustafa, S. A. Madani, and M. K. Khan, "Simdom: A framework for simd instruction translation and offloading in heterogeneous mobile architectures," *Transactions on Emerging Telecommunications Technologies*, pp. e3174–n/a, 2017, e3174 ett.3174. [Online]. Available: http://dx.doi.org/10.1002/ett.3174

[19] M. Golkarifard, J. Yang, A. Movaghar, and P. Hui, "A hitchhiker's guide to computation offloading: Opinions from practitioners," *IEEE Communications Magazine*, vol. 55, no. 7, pp. 193–199, 2017.

[20] J. Li, K. Bu, X. Liu, and B. Xiao, "Enda: Embracing network inconsistency for dynamic application offloading in mobile cloud computing," in *Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing*, ser. MCC '13. New York, NY, USA: ACM, 2013, pp. 39–44. [Online]. Available: http://doi.acm.org/10.1145/2491266.2491274

[21] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, "Cosmos: Computation offloading as a service for mobile devices," in *Proceedings of the 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. MobiHoc '14. New York, NY, USA: ACM, 2014, pp. 287–296. [Online]. Available: http://doi.acm.org/10.1145/2632951.2632958

[22] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, "A context sensitive offloading scheme for mobile cloud computing service," in *Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing*, ser. CLOUD '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 869–876. [Online]. Available: https://doi.org/10.1109/CLOUD.2015.119

[23] Z. Zhou, H. Zhang, L. Ye, and X. Du, "Cuckoo: flexible compute-intensive task offloading in mobile cloud computing," *Wireless Communications and Mobile Computing*, vol. 16, no. 18, pp. 3256–3268, 2016, wCM-16-0140.R1. [Online]. Available: http://dx.doi.org/10.1002/wcm.2757

[24] G. Xu, W. Yu, Z. Chen, H. Zhang, P. Moulema, X. Fu, and C. Lu, "A cloud computing based system for cyber security management," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 30, no. 1, pp. 29–45, 2015. [Online]. Available: https://doi.org/10.1080/17445760.2014.925110

[25] A. Graillat, M. Moy, P. Raymond, and B. D. de Dinechin, "Parallel code generation of synchronous programs for a many-core architecture," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2018, pp. 1139–1142.

[26] X. Shi, J. Liang, S. Di, B. He, H. Jin, L. Lu, Z. Wang, X. Luo, and J. Zhong, "Optimization of asynchronous graph processing on gpu with hybrid coloring model," in *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP 2015. New York, NY, USA: ACM, 2015, pp. 271–272. [Online]. Available: http://doi.acm.org/10.1145/2688500.2688542

[27] M. Essaid, L. Idoumghar, J. Lepagnot, and M. Brévilliers, "Gpu parallelization strategies for metaheuristics: a survey," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 0, no. 0, pp. 1–26, 2018. [Online]. Available: https://doi.org/10.1080/17445760.2018.1428969

[28] S. Seo, D. Lee, and K. Yim, "Analysis on maliciousness for mobile applications," in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, July 2012, pp. 126–129.

[29] M. Backes, S. Bugiel, and E. Derr, "Reliable third-party library detection in android and its security applications," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 356–367. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978333

[30] Y. Shulin and H. Jieping, "Research and implementation of web services in android network communication framework volley," in *2014 11th International Conference on Service Systems and Service Management (ICSSSM)*, June 2014, pp. 1–3.

[31] M. Lachgar, H. Benouda, and S. Elfirdoussi, "Android rest apis: Volley vs retrofit," in *2018 International Symposium on Advanced Electrical and Communication Technologies (ISAECT)*, Nov 2018, pp. 1–6.

[32] J. Sommers and P. Barford, "Cell vs. wifi: On the performance of metro area mobile connections," in *Proceedings of the 2012 Internet Measurement Conference*, ser. IMC '12. New York, NY, USA: ACM, 2012, pp. 301–314. [Online]. Available: http://doi.acm.org/10.1145/2398776.2398808